Rapport sur la campagne de mesure de performance du prototype Sirius-Delta

P. Rolin

IRIA
CENTRE DE ROCQUENCOURT

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
BP 105
78153 Le Chesnay Cedex
France
Tel. 954 90 20

Rapports de Recherche

N° 175

RAPPORT SUR LA CAMPAGNE DE MÉSURE DE PERFORMANCE DU PROTOTYPE SIRIUS-DELTA

Pierre ROLIN

Décembre 1982

RAPPORT SUR LA CAMPAGNE DE MESURE DE PERFORMANCE DU PROTOTYPE SIRIUS-DELTA

Pierre Rolin

RESUME :

Ce rapport présente les résultats complets de la campagne de mesure de performance menée sur le prototype SIRIUS-Delta. Le lecteur devra au préalable connaître les principes fonctionnels d'un système informatique réparti. Il trouvera une description de la méthode de mesure utilisée. Les coûts des primitives d'exécution répartie, de la validation à deux phases, du verrouillage, de la journalisation et des opérateurs relationnels sont exprimés en nombre d'instruction exécutées. Une méthode pour l'utilisation de ces résultats dans le but d'évaluer de futures applications réparties est proposée.

ABSTRACT :

This report gives the complete results obtained during the measurement campain realized on the prototype SIRIUS-Delta. The reader should be familiarised with the fonctional concept of a distributed system. The method used to get the result is explained. Cost of the distributed executive primitives, the two step commitment, locks, log management and of the four relational operator are expressed as a number of executed instructions. A method to evaluate a potential distributed application is proposed.

0. - INTRODUCTION

SIRIUS-DELTA est un prototype complet de base de données réparties. Sa vocation était de définir et d'expérimenter les méthodes et techniques permettant la réalisation et l'exploitation de systèmes répartis. Le lecteur intéressé trouvera dans [LEBI79] les éléments de base et toutes les références bibliographiques utiles. Le propos de ce papier étant de décrire les résultats de nos mesures, nous ne nous intéresserons ici qu'aux fonctionnalités du système réparti. En reprenant le point de vue de l'ISO [ISO227], nous pouvons dire que SIRIUS-DELTA a quatre couches au-dessus du niveau transport. La figure 1 montre les différentes couches. On distingue le niveau Global (ou producteur) qui assure l'interface avec l'utilisateur du niveau Local (consommateur) qui gère effectivement l'accès aux bases. On retrouve les 4 couches au niveau global et local.

Depuis l'utilisateur jusqu'au niveau transport :

- SGBD : c'est un système conventionnel fourni par Intertechnique sur Réalité 2000.

- SILOE : gère la distribution des données, décompose une requête utilisateur en scénarios. Les scénarios sont évalués afin de choisir le meilleur en fonction de critères tels que (la sélectivité, le nombre de transferts, la charge des machines, etc.). Un plan d'exécution réparti (PEX) est alors produit et soumis au niveau inférieur, qui apparaît donc comme étant le système d'exécution.

- SCORE : assure l'intégrité des données et la gestion des accès concurrents. Le système utilise un mécanisme contruit sur les 3 principes suivants : anneau virtuel, jeton de contrôle et séquenceur circulant pour générer un nommage unique des transactions [LEL79] appelé estampille.
  L'intégrité des données est assurée à l'aide des principes suivants, verrouillage à deux phases, évitement des étreintes mortelles et validation à deux phases [SED80].

-SER : le système d'exécution réparti fournit les services suivants :

- activation de processus distants.
- enchaînement des tâches.
- transfert des données entre processus distants.
- utilisation du parallélisme entre les différents sites.

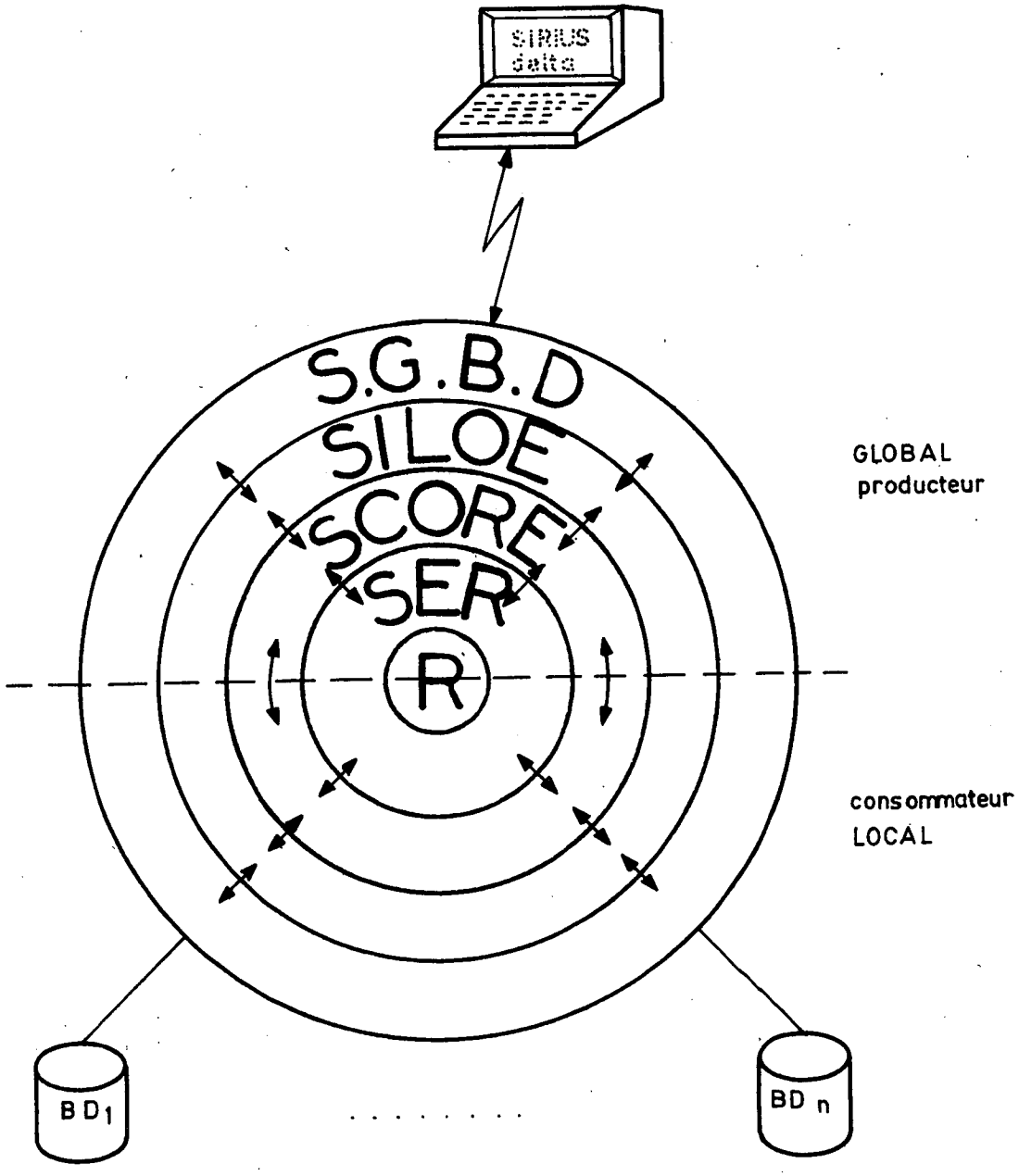


Figure 0 - Architecture de SIRIUS - DELTA

Nous nous sommes principalement intéressés à mesurer ce qui est spécifique d'un système distribué par rapport à un système centralisé. Dans ce contexte, on notera que :

- l'interface usager et l'analyse syntaxique de celle-ci est commune aux deux. Néanmoins, SILOE doit traiter la répartition et des programmes spécifiques la gère. Leur coût centralisé sur le site producteur n'a pas été mesuré dans la mesure où il dépend de la complexité de la requête.

- Chaque transaction a besoin d'un nom unique. Dans SIRIUS-DELTA, les mécanismes de l'anneau virtuel et du séquenceur circulant ont été utilisés. Le coût est discuté dans le chapitre 5.

- Le système d'exécution réparti SER a la charge de lancer les actions à distance, de les synchroniser et de permettre les transferts de données. Ses caractéristiques sont étudiées dans le chapitre 6.

- SCORE gère l'accès aux ressources. Chaque objet est verrouillé suivant un mode partagé ou exclusif et libéré uniquement en fin de transaction. Ce mécanisme n'est bien sûr pas spécifique aux systèmes distribués, il est, néanmoins, mesuré dans le chapitre 7. Le mécanisme de terminaison fiable, validation à deux phases est plus spécifique des systèmes distribués puisqu'il implique un dialogue entre producteur et consommateur. Les résultats obtenus sont discutés dans le chapitre 7. La journalisation n'est pas propre aux systèmes distribués mais, en raison de l'augmentation des possibilités de panne simple , il est impératif.

- Les opérateurs relationels utilisés n'ont rien de spécifiquement distribués, leur coût a été néanmoins mesuré afin d'avoir une idée de la répartition entre coût propre à la requête et coût induit par la distribution. Les résultats obtenus sont décrits dans le chapitre 8.

Les chapitres 1, 2, 3 et 4 décrivent respectivement les objectifs de la mesure, les outils mis en oeuvre et leurs limites, le contexte de la mesure, c'est-à-dire la configuration du prototype et les limites des machines utilisées enfin la méthode de mesure est décrite.

Le lecteur gardera à l'esprit que seuls les comportements normaux du système peuvent être mesurés. La génération de pannes, de conflits,etc. étant extrêmement difficile à réaliser systématiquement, n'a pu être étudiée.

Enfin le chapitre 9 prend un exemple de transaction et essaye de mettre en évidence où sont passés les coûts constituant cette transaction.

1. - OBJECTIF DES MESURES, CONTEXTE ET METHODE

L'existence du prototype de SGBDR SIRIUS-DELTA pose différentes questions : l'implémentation est-elle efficace ? Quels sont les coûts des différentes couches ? Au sein d'une couche quelle est la part relative d'une primitive ? Comment sont utilisés les parallélismes ? Ses effets sont-ils significativement bénéfiques ? Comment peut-on améliorer telle ou telle application ? Comment pourra-t-on définir, de manière efficace, les futurs applications ? etc.

Disons tout de suite qu'un prototype ne peut en aucun cas être utilisé pour mesurer une application, premièrement parce qu'aucune application en exploitation n'existe sur celui-ci, en second lieu, il n'a pas été conçu en vue d'une telle application mais dans le but de prouver la faisabilité d'un tel système. Nous avons donc volontairement limité nos objectifs à la mesure des éléments constitutifs principaux du SGBDR. Nous sommes convaincus que ces résultats pourront être utilement utilisés pour valider des modèles et simulations qui s'attacheront à décrire un type d'application particulière. Il aurait, par exemple, été extrêmement difficile de simuler, de façon réaliste, le comportement d'utilisateurs devant un terminal, son temps de réflexion, la complexité de ses questions, etc. Rappelons-le encore de telles mesures ne pourront être faites que par les utilisateurs des futurs SGBDR. Nous nous sommes donc volontairement écartés de la démarche traditionnellement utilisée dans les mesures [ROL79].

Les questions auxquelles nous avons cherché à répondre sont donc les suivantes :

- Comment SIRIUS-DELTA fonctionne-t-il et à quel coût ?

- La gestion du système distribué est-elle suffisamment "bon marché" pour ne pas masquer les avantages que nous attendions du parallélisme.

- Quels sont les temps de réponse et comment se décomposent-ils ?

Bien évidemment, les réponses à ces questions doivent être suffisamment générales pour pouvoir être utilisées par d'autre. Il nous a donc fallu trouver un moyen d'obtention de résultats qui soit aussi peu dépendant que possible des machines sur lesquelles notre prototype est implémenté. Il nous a semblé que le nombre d'instructions exécutées était le critère le plus sûr. En effet, le langage de programmation joue sur le taux d'expansion du code et donc sur la quantité de code produit et exécuté. Si l'on dispose d'une bonne estimation de ce taux d'expansion, on pourra en tenir compte pour évaluer les coûts dans un autre langage. De même, si l'on dispose d'une machine plus ou moins puissante, on pourra estimer un taux d'expansion de l'une à l'autre. Les courbes obtenues en nombre d'instructions sont assez indépendantes du taux de défaut page, ce qui n'est pas le cas du temps de réponse.

Le dernier problème consiste à séparer les coûts relatifs de système utilisant les fonctionnalités de systèmes inférieurs . Nous avons, pour cela, cherché à utiliser une méthode de mesure ascendante : coût des éléments des couches basses, puis coût des systèmes utilisant ceux-ci par une méthode ressemblant à une double pesée.

La répétitivité d'une mesure dans des conditions identiquesest pour nous un facteur essentiel de bonne conditions de travail. Nous avons donc figé le contexte de mesure.

2. - OUTILS ET METHODE DE MESURE.

Notre premier souci a été de disposer d'un appareillage de mesure qui nous permette d'observer le maximum de paramètres caractérisant la charge des machines, le temps de réponse etc. Pour cela, nous avons utilisé concurremment deux outils, un moniteur de mesure logiciel et un moniteur de mesure matériel.

2.1. - Moniteur de mesure logiciel

Il est basé sur la prise d'événements qui nous permettent de calculer le temps écoulé pour l'exécution d'une fonction, par exemple, la pose des sondes qui captent ces événements est réalisée en utilisant un modèle décrivant les algorithmes, fonctions ou primitives [ROL80]. Les informations ainsi captées nous permettent de calculer le temps écoulé entre deux événements, la fréquence d'arrivée des événements, l'écart-type de ces valeurs, la taille des files d'attente éventuelles, etc. Par contre, le moniteur logiciel fonctionne séparément sur chaque machine. Comme dans un système distribué on ne dispose pas d'une horloge unique, il n'est pas possible ainsi d'étudier des événements produits sur deux machines distinctes et donc le parallélisme. Tout moniteur logiciel modifie le comportement du système observé puisqu'il consomme des ressources pour s'exécuter. Nous ne l'utiliserons donc jamais en même temps que le moniteur matériel lorsque nous voulons une mesure précise en nombre d'instructions. Par contre, il est toujours possible de répéter deux fois la même mesure, une fois avec le moniteur logiciel actif et une fois sans, tout en sachant que les durées observées sont allongées.

2.2. - Le moniteur matériel

Il s'agit d'un moniteur Testdata MS88. Les sondes ont été posées sur les 3 machines profitant ainsi de leur localisation physique dans une même salle. La centralisation de ces sondes sur le moniteur matériel permet d'observer le parallélisme entre les 3 machines et au moins d'extraire des mesures pendant les mêmes intervalles de temps. Malheureusement, les dépouillements disponibles sur cet outil ne nous permettent d'obtenir que des résultats statistiques, or nous aurions apprécié la possibilité de tracer, par exemple, les courbes d'activités CPU sur les 3 machines avec une période de l'ordre de quelques secondes.

Celles-ci nous auraient permises de voir la décomposition des activités (parallélisme) CPU en fonction des travaux. La période minimale étant la minute, les effets instantanés sont complètement lissés. Nous pourrons seulement commenter les effets vus sur le bargraph dont la période d'affichage est de cinq secondes.

3. - LE PROTOTYPE

SIRIUS-DELTA est implémenté sur trois machines Réalité 2000, dotées d'une mémoire de 256 k octets à tore, 2 disque Ampex M1717-1 chaînés, 8 à 12 voies asynchrones pour terminaux, 2 coupleurs parallèles bidirectionnels M1612, utilisés pour réaliser les réseaux. Les machines sont dotées d'un mécanisme de mémoire virtuelle s'étendant sur tout l'espace disque disponible. Leur jeu d'instruction est orienté traitement de caractères. Les langages disponibles sont: l'assembleur dans lequel le minimum nécessaire de programmation a été réalisé et un basic interprété dans lequel la grande majorité des programmes ont été écrits. Le choix de basic a répondu pour nous bien sûr à un impératif de production rapide de logiciel et bien sûr la qualité de l'implémentation s'en ressent.

La pose des sondes sur ce minicalculateur de deuxième génération s'est avérée assez difficile. Il nous a fallu trouver les points du microprogramme correspondant à :

- début de traitement d'une instruction,
- passage en moniteur,
- passage en mode utilisateur,
- inactivité (Idle).

En fait ce que nous avons obtenu est assez imprécis en ce qui concerne le temps passé en moniteur. En effet, le traitement des interruptions se fait sans changer de mode et il n'a pas été possible d'extraire un signal fin d'interruption. On comptabilise donc, en mode virtuel, des travaux faits en fait pour le moniteur, par exemple, le traitement de l'horloge, fin de traitement des E/S disque, etc. Bien que nous ayons dans le dépouillement séparé les instructions exécutées en mode virtuel des instructions exécutées en mode moniteur, il nous est, de ce fait, impossible d'exploiter ce résultat.

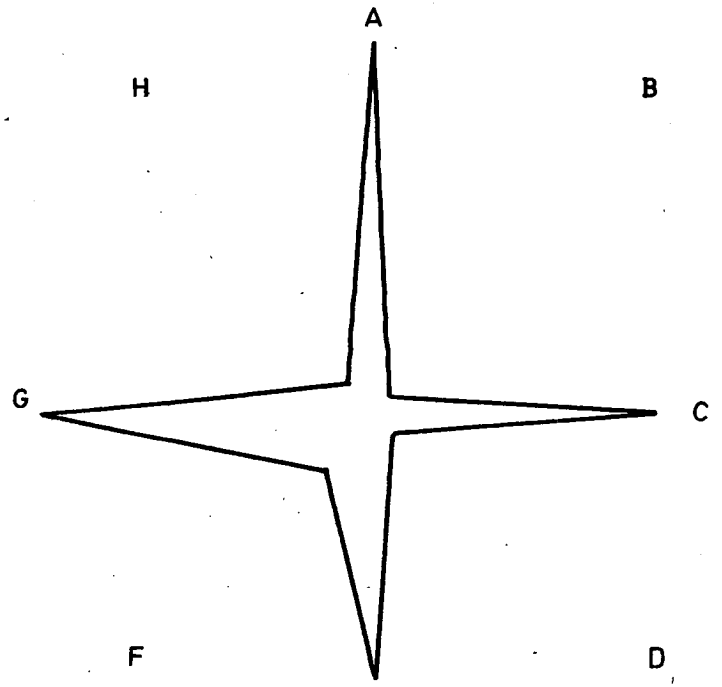
La caractéristique essentielle de cette machine est d'être assez lente (faible en CPU) comme le montre la figure 1 et le tableau associé et qui ont été obtenus à partir d'un benchmark du constructeur censé refléter une application typique Base de données. Le nombre maximum d'instructions exécutées par seconde est voisin de 25 700, soit environ 39 µs par instruction. Différentes mesures ont montré que le nombre d'instructions exécutées par seconde pouvait varier de 18 000 pour un programme basic exécutant beaucoup de mouvement de chaîne à 28 000 pour un programme assembleur système. Cette lenteur caractéristique de nos minis sera, en général, la cause des temps de réponse élevés. Elle se traduira toujours par une très forte charge du CPU.

4. - METHODE DE MESURE

Nous avons cherché à mesurer les coûts intrinsèques de chaque fonction de SIRIUS-DELTA. Pour ce faire, il nous fallait évaluer le bruit de fond dû au système réparti et non comptabilisable à une primitive. Bien sûr, lorsqu'une primitive est activée, elle peut limiter le bruit de fond puisque le système pourra alors éviter l'exécution de fonctions d'entretien, activées au passage par la primitive. Il ne pourra en être tenu compte avec précision mais cela ne modifie probablement pas significativement la forme des courbes et, de toute façon, correspond à la nature du travail exécuté.

La première mesure à faire consiste donc à mesurer le bruit de fond moyen : A (figure 2). Celui-ci connu est considéré ultérieurement comme constant. Dans SIRIUS-DELTA, le bruit de fond est constitué par :

- la gestion de la mémoire virtuelle (spécificité R2000) et des tâches ;
- la gestion des liaisons et ligne, réalisée par deux processus sur chaque machine ;
- la gestion de l'anneau virtuel.



14 PROCESS 02/10/82

ET= 3590.000 SEC FROM=12:30:03 TO 13:29:53

	ID= 6A		VALUE	PCT
B=$\overline{A}$	A	1= CP BUSY M1	346069.	96.397
	G	2= MODE VIRTUEL M1	310885.	86.597
	H	3= MODE MONITEUR M1	35211.0	9.8080
		4= ACT DISQUE 0 M1	129714.	36.132
		5= ACT DISQUE 1 M1	227698.	63.425
		6= EMISSION VERS M3	0	0
		7= EMISSION VERS M2	0	0
		8= NCP ET OU DISQUES M1	12579.0	3.5038
		9= NCP ET OU EMIS M1	0	0
		10= OU DIS ET OU EMIS	0	0
	C	11= OU DISQUES M1	281504.	78,143
	D	12= ET DISQUES M1	75908.0	21.144
	F	13= CP ACT ET DISC INAC	77144.0	21.488
	E	14= CP ACT ET OU DISC ACT	268925.	74.909

Tableau 1

Figure 1 : Kivigraphe - caractéristique d'une machine trop faible en CPU extrait du tableau 1

La phase suivante consiste à exécuter la primitive la plus basse du système. Il faut bien sûr s'assurer que celle-ci n'utilise rien d'autre. Si cette primitive a plusieurs paramètres, on fera varier,un par un, ces différents paramètres, de manière à déterminer les coûts propres associés à chacun d'eux. Une telle démarche utilise les principes d'une architecture à couche.

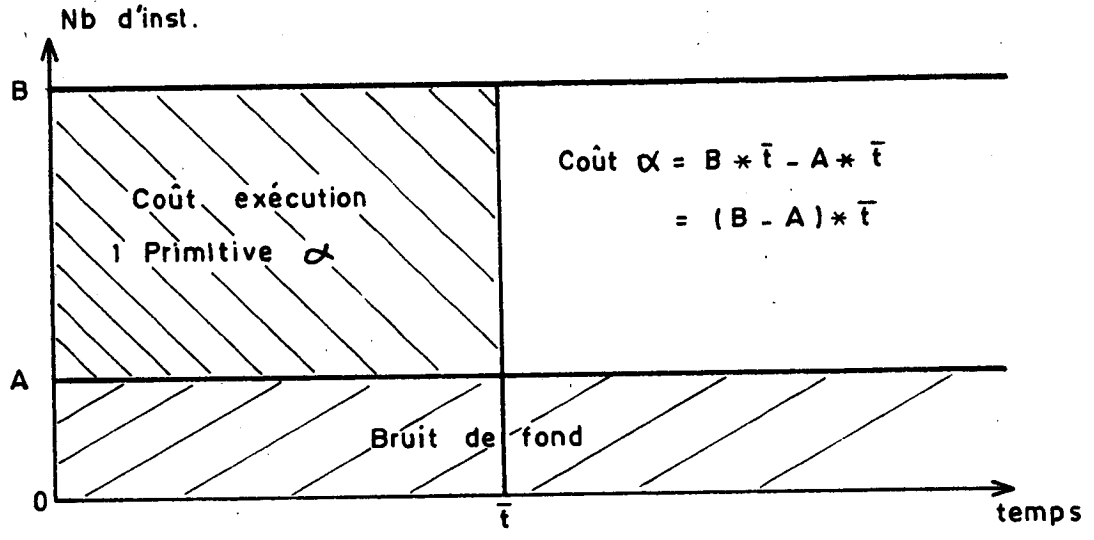


Figure 2 - Coût d'une primitive α de temps moyen d'exécution $\bar{t}$

La figure 2 explique comment calculer le coût d'une primitive α. Le moniteur matériel nous donne le nombre moyen d'instructions B exécutées par seconde pour l'éxécution de cette primitive. Il est, bien évident, que, pour que cette moyenne ait un sens, il faut lancer cycliquement α pendant une période assez longue. Nous utilisons cette méthode de lancement cyclique pour trouver le temps moyen d'exécution de la primitive $\bar{t}$. Ce temps moyen permet de calculer le coût de la primitive de la manière suivante :

$$\text{coût } (\alpha) = \bar{t} * (B-A).$$

Lorsque α est doté de paramètres, nous tracerons la courbe coût $(\alpha_{(p)})$ où p prendra des valeurs variables du paramètre.

5. - OVERHEAD SIRIUS-DELTA

L'overhead ou bruit de fond dans SIRIUS-DELTA se compose des fonctionnalités permanentes nécessaires au bon fonctionnement du système distribué. Elles ne peuvent être comptabilisées pour l'une ou l'autre des primitives :

- gestion de la mémoire virtuelle, cette caractéristique des R2000 provoque des entrées-sorties disque pour la mise à jour sur les disques des pages modifiées en mémoire. L'accès a une page absente.
- horloge (gérée par micro-programme) comme toutes les interruptions elle consomme du temps CPU.

Les deux caractéristiques sont propres au système R2000 et n'ont pu être explicitement comptabilisées puisque nous n'avons pas pu disposer de 2 sondes début et fin interruption. En outre, le taux de défaut de page dépend beaucoup de la charge de travail demandée, de la quantité de code des programmes et du nombre d'accès aux fichiers. Le taux d'écriture sur disque est dépendant de ces mêmes critères. Pour plus de précision sur le fonctionnement de la mémoire virtuelle, le lecteur se reportera à [INT1].

La gestion des lignes est une caractéristique propre à un système distribué . Nous avons implémenté une mini-station de transport assurant la détection des pannes, le reroutage des messages et la gestion des liaisons logiques. Pour ce faire, un mécanisme de bulles et de temporisation est utilisé. Son coût a pu être évalué en laissant la configuration tourner à vide pendant une période de temps suffisamment longue et sans pannes.

Le nombre moyen d'instructions exécutées par seconde est de 2600, représentant environ 14.5 % de l'activité du CPU. L'activité des lignes est de 0,6 %, soit environ 0,13 bloc par seconde. Il apparaît ici très clairement que le coût de gestion des messages est prohibitif sur cette machine. Ceci est confirmé par les mesures faites sur la station de transport (ST). Pour émettre un message d'un processus à un autre processus,

il faut au minimum 3 messages de niveaux inférieurs (cf. figure 3). Rappelons que notre ST correspond à un niveau 4 ISO.

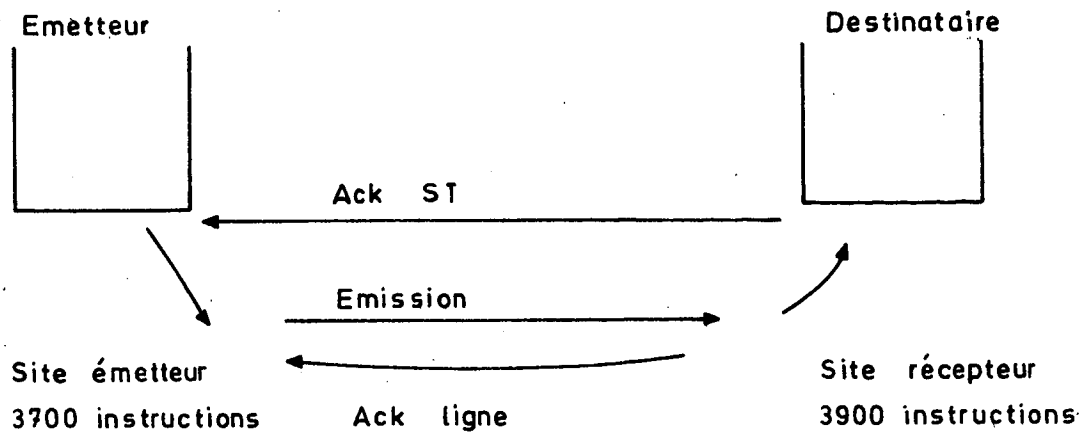


Figure 3 - Protocole d'émission d'un message

Le nombre maximal de messages par seconde est de 6, soit 3000 caractères par seconde (le bloc de base émis étant d'une longueur maximale de 500 caractères). Nous n'avons pas essayé de mesurer des messages de longueur supérieure à 500 caractères. Le coût global d'un message est de 7 600 instructions dont 3700 sont exécutées sur le site émetteur et 3900 sur le site récepteur. La mesure montre, par ailleurs, que :

- le CPU émetteur atteint 90 % d'activité (l'attente sur les disques n'est que de 4 %).
- le CPU récepteur atteint 96,5 % d'activité (pas d'attente disque).

Il est donc bien clair que le débit des messages est limité uniquement par la faible puissance de nos CPU. Ceci est confirmé par le taux d'activité de la liaison qui atteint en moyenne 26 % et au mieux de façon instantanée 30 %. Ce très faible débit écoulé sur les lignes provoquera des attentes pour nos processus mais puisque les CPU saturent en exécution, nous ne pourrons pas observer les attentes CPU sur les lignes. On trouvera dans [MES I006] des éléments d'information sur la durée effective (temps écoulé) pour acheminer un message. La mesure hardware montre qu'il ne faut pas espérer mieux que 166 millisecondes.

Etant donné que nos lignes sont capables de transférer 50000 caractères par seconde, la durée de transfert d'un message vient principalement du coût de traversée des couches logicielles. Avec pour conséquence une consommation importante de la ressource CPU. Il est évident qu'un processeur frontal améliorerait les choses de ce point de vue. Il n'est pas certain, par contre, qu'il apporterait une réduction importante du temps de transfert.

Nous insisterons encore une fois sur le coût élevé des traitements dû à la soumission et émission, réception et prise en compte d'un message.

L'anneau virtuel et le séquenceur circulant est un élément d'overhead inhérent au système distribué. Le lecteur intéressé trouvera dans [LEL 79], [ROL 80] et [SED 81] les motivations et une description détaillée de ce mécanisme. Le coût de l'anneau virtuel est de 4500 instructions par seconde, soit 21 % du CPU dont il faut déduire 2600 pour la gestion de ligne, soit 1900 instructions par seconde.

La vitesse de rotation du séquenceur circulant est pour les 3 machines d'environ 18 s. Le délai est principalement dû à une temporisation de 5 secondes installées sur chaque machine afin d'éviter la saturation des machines uniquement a cause du séquenceur. Ceci étant dû principalement à la faible puissance des CPU et au petit nombre de sites qui provoque une rotation "rapide". La figure 4 montre les coûts relatifs de la gestion de ligne et du séquenceur.

L'activité des lignes est inférieure à 1 %, soit une augmentation de moins de 0,4 % par rapport à la gestion de ligne. Avec la temporisation introduite, le séquenceur produit une surcharge réseau très modérée.

Le coût des pannes n'a pu être mesuré en nombre d'instructions du fait de la difficulté de reproduire le phénomène cycliquement.

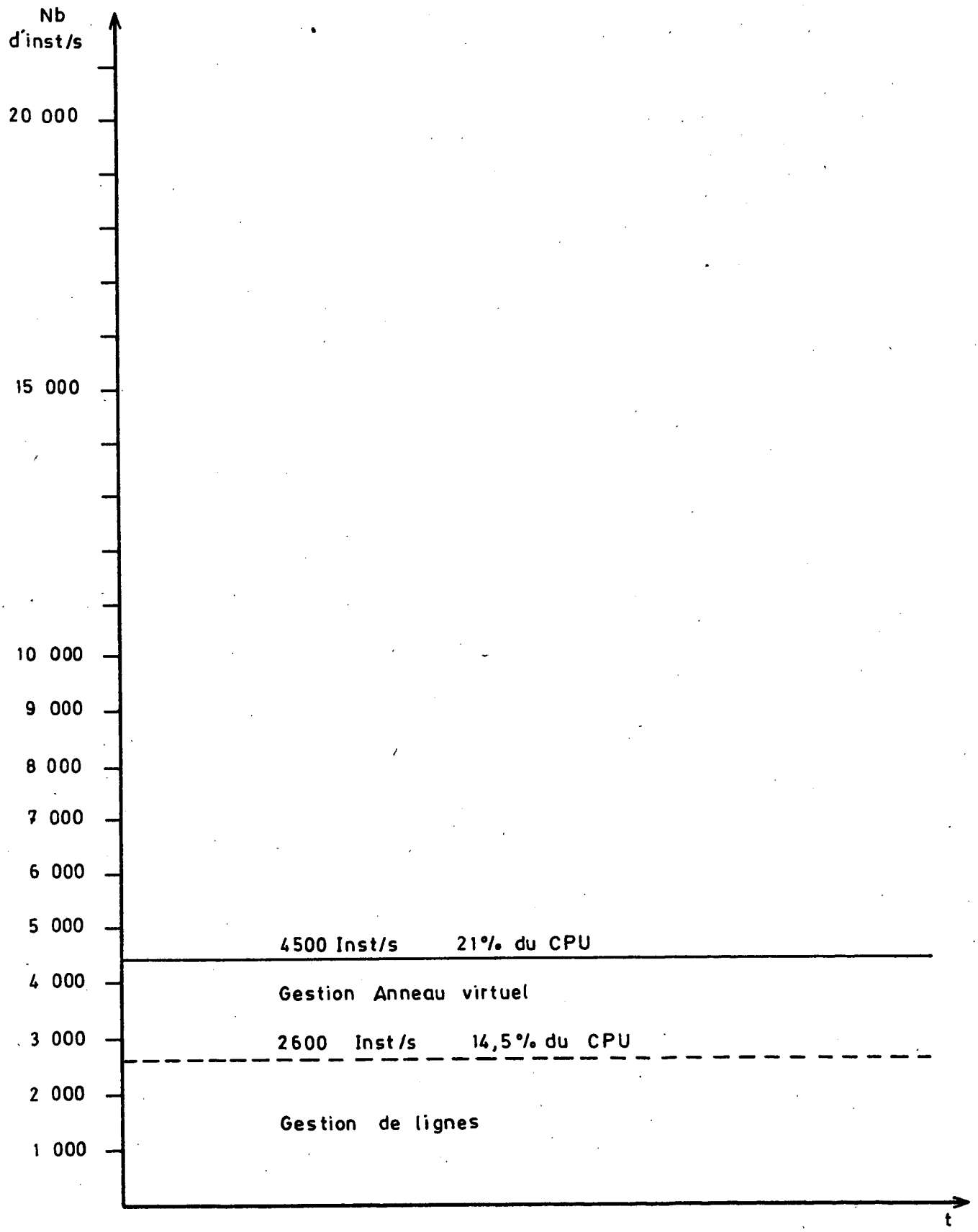


Figure 4 - Coût relatif de la gestion de ligne et de l'anneau virtuel

6. - LE SYSTEME D'EXECUTION REPARTI SER

Les fonctionnalités de SER sont décrites succinctement par les points ci-dessous. Le lecture intéressé trouvera dans [TRE 79] une description détaillée de celle-ci.

A) Activer un programme sur un processeur quelconque.

B) Enchaîner les différents programmes locaux suivant les modalités spécifiées par l'application pour le compte d'un même utilisateur.

C) Assurer les transferts de données nécessaires entre deux programmes s'exécutant pour le même utilisateur.

D) Exprimer sous forme logique les conditions d'activation d'un programme, fonction éventuelle des résultats d'autres programmes.

E) Utiliser les possibilités de parallélisme offertes par la machine multiprocesseurs que constitue l'ensemble des processeurs interconnectés.

F) Détecter et signaler les pannes de processeurs.

G) Signaler les fins normales ou anormales des programmes.

Les points F et G n'ont pu faire l'objet d'une mesure particulière. Le point E n'est pas réellement quantifiable sur un prototype limité où l'activation des processus sur une machine est prédéterminée.

Le concept de base de SER est le Plan d'Exécution Réparti PEX. Le PEX contient un ensemble d'actions locales qu'il faut activer à distance. Chaque action locale possède :

- un certain nombre de variables de synchronisation qu'il faut mettre en place puis activer ;
- des fichiers de données temporaires FDT qu'il faut mettre en place puis transférer.

Chacun de ces éléments a pu être considéré isolément. Il est bien évident qu'ici le phénomène d'empilement des couches devient important et, de ce fait, joue sur la précision de la mesure.

6.1. - Coût des Actions Locales : AL

Le coût d'une AL se décompose en deux éléments : le coût de la prise en compte du PEX sur le site global (figure 5) et son coût d'installation sur le site local. L'installation et le lancement d'une Action Locale se fait en deux phases :

- envoi au site local concerné de l'AL. Celui-ci crée un contexte pour celle-ci puis envoie un acquittement sur site global.

- lorsque toutes les AL ont été acquittées, le site global envoie pour chacune d'elle un ordre d'exécution. Le site local exécute l'AL lorsqu'il dispose de toutes les ressources, que les conditions d'activations sont prêtes et qu'il dispose d'une ressource processeur. A la fin de l'AL, un message de fin d'AL est envoyé au site global.

Le PEX est terminé lorsque le site global a reçu toutes les fins d'activation.

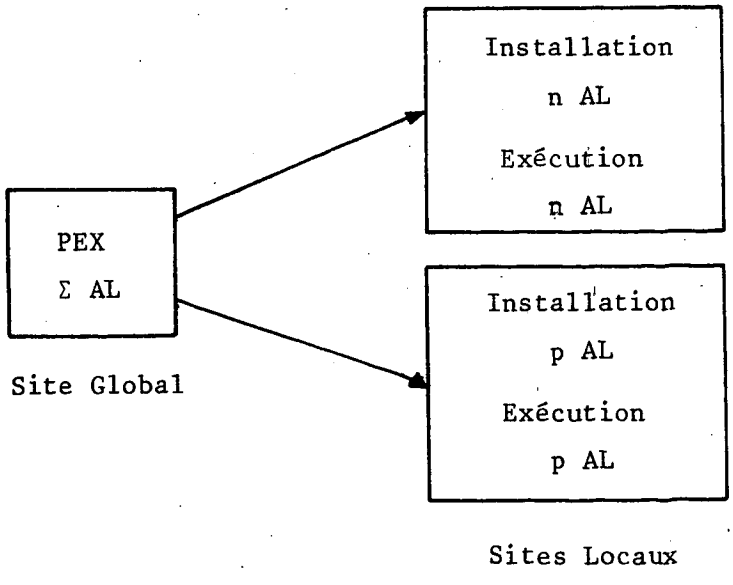


Figure 5 - Schéma fonctionnel de SER

Mesurer le coût minimal d'une AL consiste à créer des PEX qui ont n AL à exécuter sur chaque site. Chaque AL est vide, c'est-à-dire qu'elle n'a aucune condition d'activation (lancement immédiat), aucun FDT et ne fait rien (retour immédiat).

Le coût sur le site global apparaît de la manière suivante un coût d'entrée dans le programme plus un coût de lancement proprement dit. La courbe de la figure 6 montre que nous obtenons une droite d'équation :

Coût n AL = n * 37 500 Instructions + 10 000 Instructions

Les 10 000 instructions correspondant au coût d'entrée dans le programme. Le coût de lancement distant d'une AL correspond à 37 500 instructions. En nombre d'instructions, nous ne pouvons pas décomposer entre la phase initiale et finale. La Figure 7, obtenue à partir du moniteur logiciel, permet de voir les temps relatifs à ces deux parties.

Sur le site local, les AL sont reçues une par une. Il apparaît que le coût d'installation et de lancement d'une AL est de 45 000 instructions. (fig. 6)

Si l'on se rappelle que le processus de lancement d'une AL comporte 4 messages :

	Global	Local
Création AL CTXALR	envoi	reçu
ACK création AL CTXALA	reçu	envoi
Lancement AL CTXALL	envoi	reçu
Fin AL CTXALF	reçu	envoi
soit en nombre d'instructions :	15 200	15 200

Donc dans le lancement d'une AL, 40 % des instructions (15 200 instructions) sont dues aux messages et 60 % (22 300 instructions) à leur interprétation sur le site Global.

Sur le site local, 34 % des instructions sont dues aux messages et 66 % à la mise en place de l'AL puis son lancement.

Le temps de réponse pour la mise en place d'une PEX n'est pas inférieur à 18 secondes car nous avons utilisé des transactions précataloguées pour lancer nos PEX. Donc chaque transaction a besoin pour s'exécuter d'un ticket et, de ce fait, attend le séquenceur circulant dont la période de rotation est environ 18 secondes. On s'aperçoit qu'à partir de 6 AL par PEX, le temps de réponse augmente au-delà de la période de rotation du séquenceur. A l'inverse, le taux d'activité du CPU Global augmente jusqu'à cette valeur et se stabilise a une valeur proche de 80 % en moyenne. Il y a donc saturation du CPU global. Les CPU des sites locaux suivent une courbe voisine et se stabilisent vers un taux d'activité de 50 % donc non saturés, ils sont en attente de soumission de nouvelles AL. L'activité des lignes croît aussi puis se stabilise à la même hauteur à un taux d'activité proche de 12 %. On remarque, en outre, que le taux d'activité simultané des deux lignes C .1 et C .2 est élevé (25% de l'activité) exhibant l'utilisation du parallelisme.

Cette mesure fait donc apparaître une saturation du CPU global lors du lancement d'un PEX. Cette saturation pourrait être évitée en groupant toutes les AL destinées à un même site dans un seul message. Par exemple, pour un PEX de 14 AL, destinées à deux sites, il ne faudrait pour le site global n'envoyer et recevoir que 4 messages au lieu de 14. Le coût de lancement d'un PEX deviendrait alors proportionnel au nombre de site à atteindre et non au nombre d'AL. Sur le site local, il ne faudrait plus recevoir et envoyer que deux messages, par contre, il faudrait toujours, bien sûr, mettre en place et lancer en exécution les AL reçues. Le coût aurait l'allure suivante si l'on prend pour coût d'installation d'une AL, environ 29 800 instructions (en y incluant le coût d'entrée dans le programme) :

$$\text{Coût (inst) de n AL sur site local} = n \times 29800 + \underbrace{15\,200 + K}_{\text{constant}}$$

où K est une constante représentant le coût d'entrée dans le programme et 15200 Inst le coût des receptions/emissions de messages.

La figure 7, obtenue à partir du moniteur logiciel, fait ressortir combien l'attente des messages est importante dans la donnée d'exécution d'une AL. Sur le site local on s'aperçoit qu'une AL, dont la durée totale est de 3,8 s, en consacre seulement 0,3 s à se mettre en place, 0,45 s a s'exécuter (AL vide), il s'agit donc uniquement du coût de scheduling, alors que 2,1 s se trouvent dans les délais entre le message

PEX doté de n AL n/2 par site local	Site global		Sites locaux					
			1		2			
	Temps de réponse de la transm.	Activité CPU	Activité CPU	Activité ligne A Global <-> 1	Activité CPU	Activité ligne Global <-> 2	Les 3 CPU Actifs	Activité simultan des lignes Global <-> 1 et Global <-> 2
1	18 s	51 %	33,5 %	5,3 %	20 %	1 %	3,7 %	0,1 %
2	19,1 s	58 %	29 %	5,3 %	32 %	5,4 %	11,6 %	0,9 %
4	21 s	68 %	40 %	7,5 %	36 %	8,5 %	20,1 %	1,6 %
6	22 s	89 %	50 %	12,1 %	46 %	12,1 %	34,8 %	3,1 %
8	28 s	77 %	48 %	10,1 %	44 %	11,6 %	31,5 %	2,7 %
10	37,5 s	72 %	46 %	10,5 %	42 %	11,1 %	29,8 %	2,6 %
14	42,7 s	82 %	51 %	12,6 %	47 %	13,2 %	35,5 %	3,2 %

Tableau 1 - Activité comparée du site global et des sites locaux

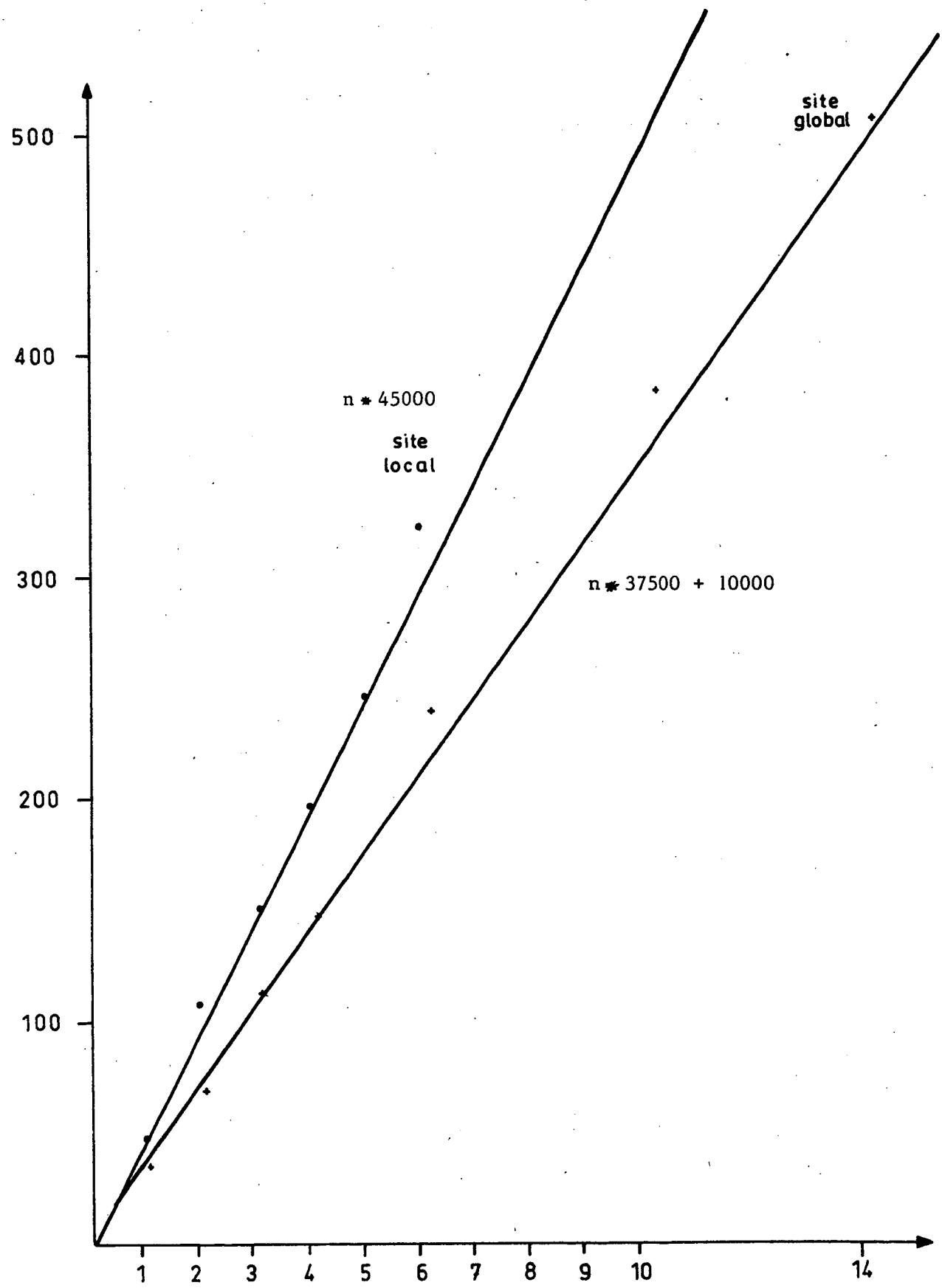


Figure 6 - Coût d'installation des AL - 1 AL ≈ 45 000 instructions

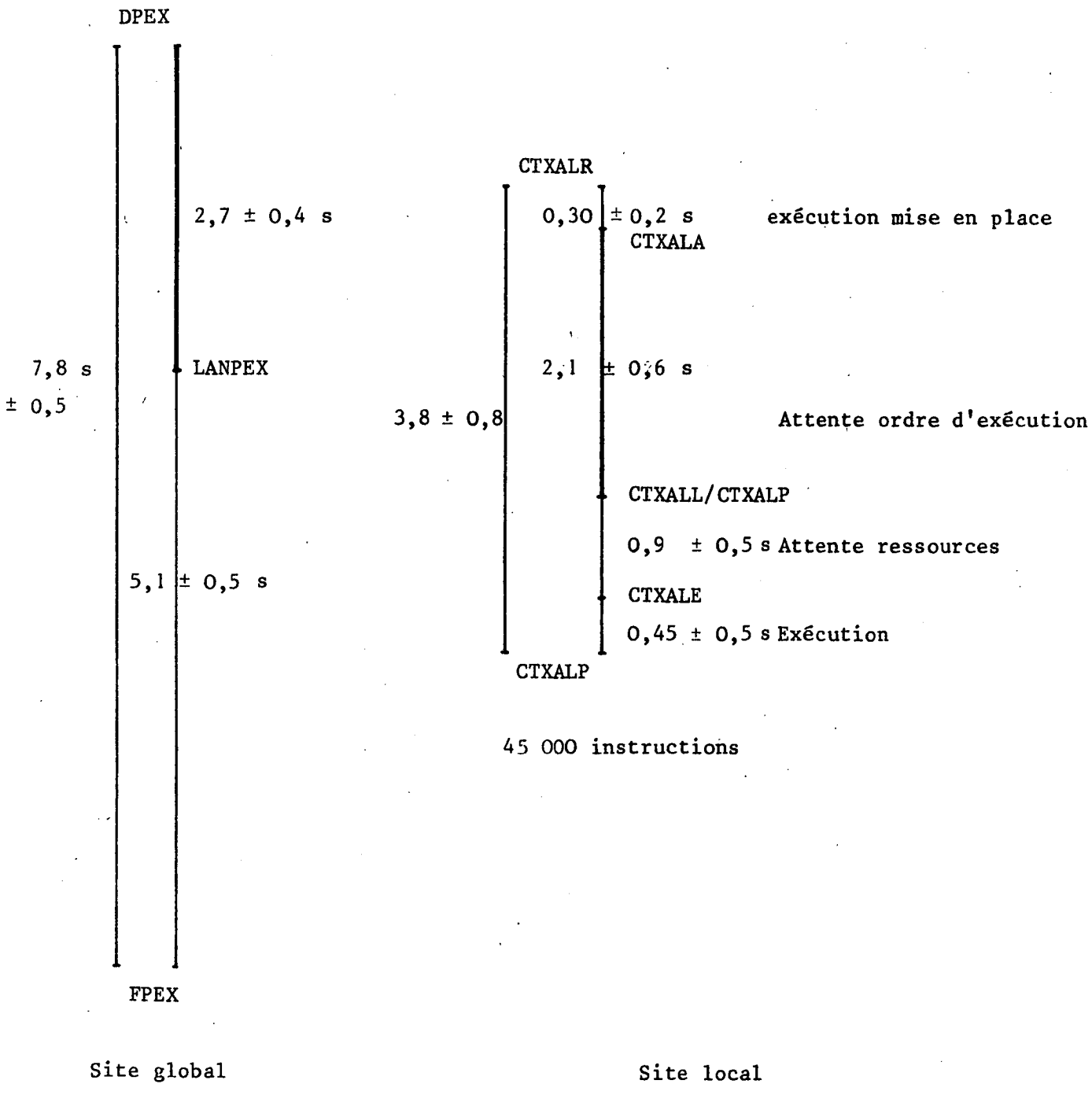


Figure 7 : PEX avec 3 AL, sans CA ni FDT, fréquence des PEX : 35 s

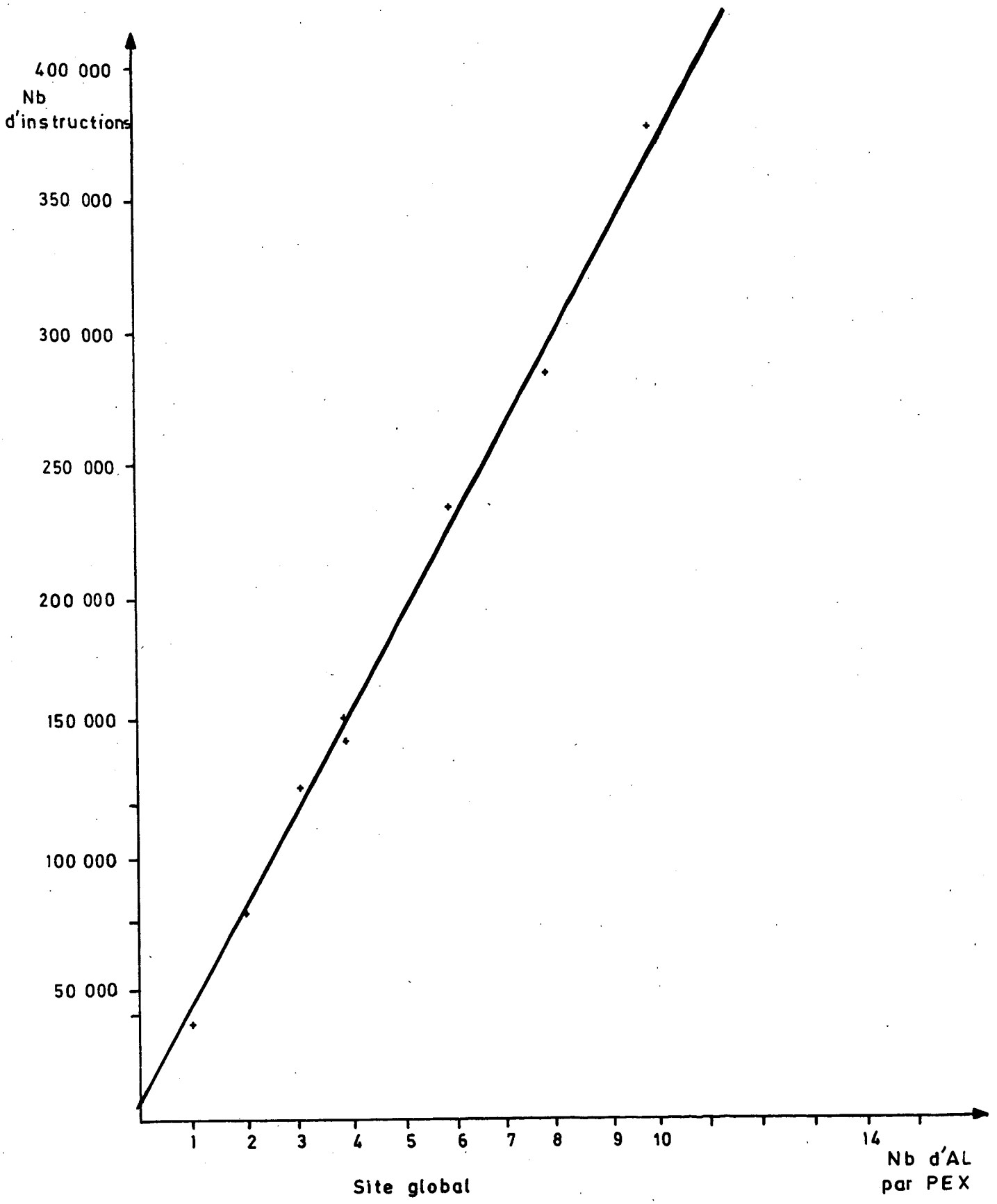


Figure 8 - Nombre d'instructions pour le lancement d'un PEX avec n AL

Coût installation n AL = 10 000 instructions + n * 37 500 instructions
en nombre d'instructions

acquittement AL et lancement AL, 0,9 s pour l'attente d'une ressource processeur (file d'attente). Sur le site global, on s'aperçoit que les durées sont encore allongées puisqu'il y a les durées de transfert de deux messages en parallèle mais traité séquentiellement bien sûr sur le site global.

6.2. - PEX avec des variables de synchronisation VS

Pour pouvoir mesurer le coût des variables de synchronisation, nous avons mis en oeuvre des PEX à l'allure suivante (figure 9). Une AL_1 active n variables de synchronisation sur une AL_2. Les deux AL sont sur des sites distincts afin d'avoir les messages pour les transmissions de synchro et de pouvoir distinguer la partie activation et prise en compte de la synchronisation.

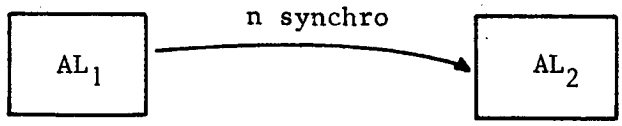


Figure 9 - Schéma de synchronisation de 2 AL

Chaque variable de synchronisation est traitée individuellement et fait donc l'objet d'un message.

Sur le site qui active les VS, on observe une droite (figure 10).

Activation n VS = n * 6000 + 32 000 instructions.

Le coût d'entrée dans le programme d'activation est élevé mais le coût d'activation distante est très faible 6000 instructions, si l'on se rappelle qu'il faut envoyer un message dont le coût sur le site émetteur est 3700 instructions soit plus que la moitié. Il faut donc souligner les bonnes performances de ce mécanisme.

Nous n'avons pas vérifié que l'activation de VS à divers instants de l'AL restait linéaire. On peut, en effet, activer une VS en début d'AL (avant son lancement) ou en fin d'AL (après son exécution). Nous aurions dû logiquement obtenir une courbe du type p * 32000 instruc-

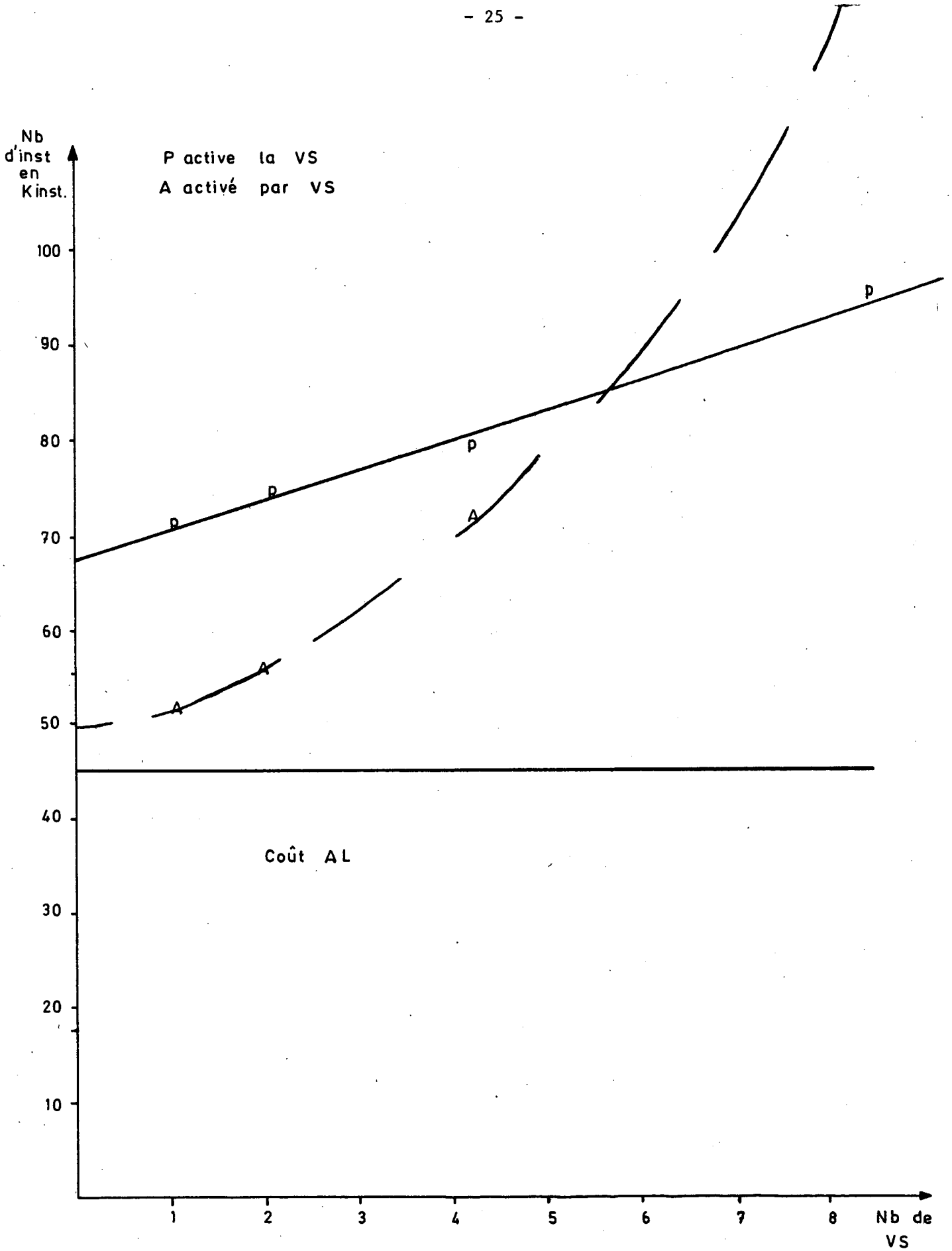


Figure 10 - Coûts des activations de VS . Activation n VS = 32 000 + n * 6 000

tions où p est le nombre de fois où est invoquée la fonction d'activation des synchronisations auquel il faut ajouter n * 6000 instructions,n étant le nombre de VS invoquées dans cet appel.

Sur le site récepteur, la courbe n'est plus une droite, ce qui peut paraître surprenant. En fait, à chaque fois qu'une VS est activée, le programme regarde s'il peut ou non lancer l'AL. Il y a donc un travail qui varie de la manière suivante. Si q est le nombre d'instructions pour tester si une VS active ou non, le coût de réception d'un message est de 3900 instructions et p le coût d'activation de la VS indiquée dans le message.

$$\text{Coût pour n VS} = n * (3900 + p) + q * \left(\sum_{i=1}^{n} i \right).$$

Nous n'avons pas vérifié si l'ordre d'activation avait de l'effet sur cette loi. Cela est très probable puisque le programme arrête les tests dès qu'il trouve un VS non activé. Or notre programme de mesure active les VS dans l'ordre croissant. Il faudra donc se méfier pour une modélisation des effets du à une activation aléatoire de VS alors que le programme les teste toujours dans le même ordre.

L'évaluation de p et q peut être faite en prenant des points de la courbe :

$$n = 2 \quad 2 * 3900 + 2p + q \sum_{i=1}^{2} i = 12\,000$$

$$n = 8 \quad 8 * 3900 + 8p + q \sum_{i=1}^{8} i = 70\,000$$

En résolvant ce système d'équations, nous trouvons p = 2900 instructions et q = 433,3 instructions. Evidemment,il ne s'agit que d'ordre de grandeur, étant donné le petit nombre de points effectivement observés.

L'effet de boucle dû au test aurait pu être évité en maintenant à jour un compteur du nombre de VS non encore prêtes.

Le tableau 2 confirme les commentaires précédents, l'activité CPU des sites locaux reste modérée même pour un grand nombre de VS. Il faut donc souligner l'efficacité du mécanisme. Le temps de réponse global est constant puisqu'il s'agit en fait de la période de rotation du jeton.

L'augmentation de l'activité de la ligne 1 <-> 2 est strictement proportionnel au nombre de messages échangés pour mettre à jour les VS.

La figure 11 montre la décomposition en temps écoulé d'un PEX dont les AL sont synchronisés par des VS. L'événement CTXALP signifie que toutes les VS sont actives et donc que l'AL peut être activée. Il est évident qu'il ne s'agit que d'un cas de figure parmi d'autres. La phase finale de l'AL est allongée par rapport à la figure 7 du fait de l'activation d'une VS distante.

6.3. - Les fichiers de données temporaires : FDT

Ils sont l'élément clé de tout système d'exécution réparti. La réalisation soignée et optimisée des mécanismes de gestion des FDT conditionne l'efficacité du système. Nous pouvons décomposer ce coût en deux éléments :

- le coût de déclaration qui impose la création et la réservation des structures nécessaires et le coût de libération de ces mêmes ressources.

- le coût de transfert d'un article de FDT proprement dit.

Sur la figure 12, nous avons noté le principe de 2 types de mesures. La première a consisté à lancer des AL qui déclaraient un FDT en entrée et un en sortie. Sur la seconde, nous avons séparé la fonction entrée et sortie sur deux machines.

PEX doté de 2 AL n synchro.	Site global	Sites locaux						
		1		2				
	Temps de réponse en s	Activité CPU %	Activité CPU %	Activité ligne global <-> 1 %	Activité CPU %	Activité ligne global <-> 2 %	Les 3 CPU actifs %	Activité lig global 1<-> %
0 (ligne 2 du tableau 1)	19,1	58	24	5,3	32	5,4	11,6	8
1	19,2	58,7	34,4	5,3	31	5,4	10,6	1,3
2	19,2	57	35,5	5,1	32	5,2	12,5	1,8
4	18,5	57	36,4	4,9	36,4	5,3	14,8	2,9
8	19,5	58	39,2	4,9	45,3	5	15,6	5

Tableau 2 - Activité comparée du site global et des sites locaux

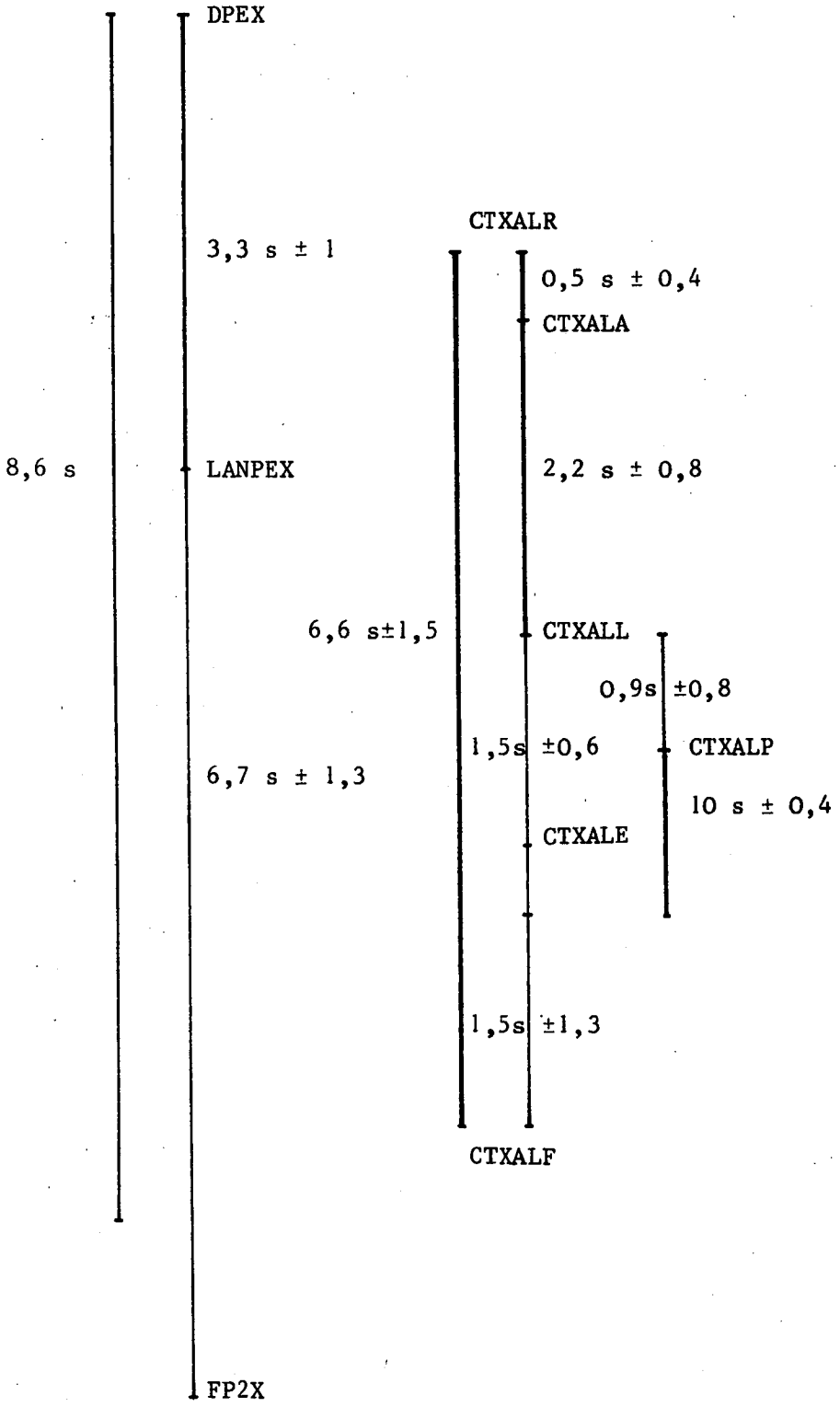


Figure 11 - PEX avec 3 AL vides avec CA, pas FDT

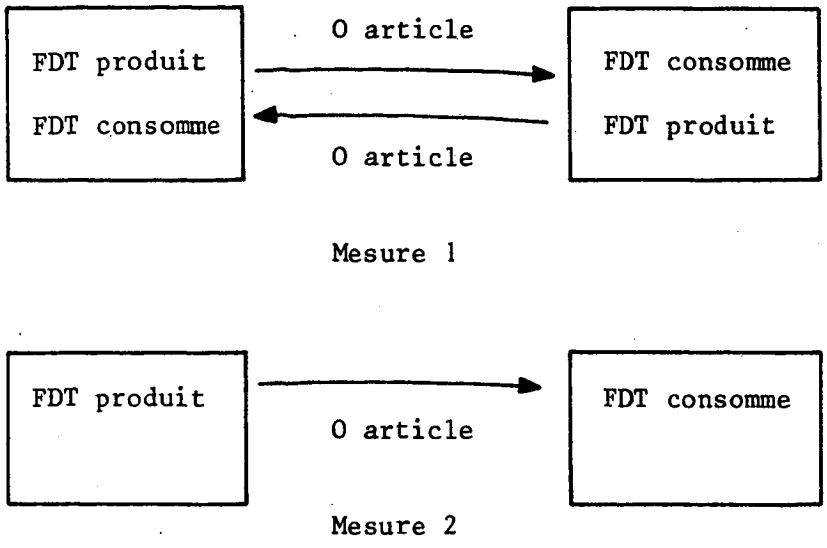


Figure 12 - Principe de la mesure sur les FDT

La figure 13 présente les résultats obtenus. Le premier problème qui apparaît est que la courbe de la mesure 1 n'est pas la somme des deux courbes de la mesure 2. Ce phénomène est dû à deux causes principales :

- l'imprécision de la mesure bien sûr,
- le fait que lorsque l'on active plus fréquemment une même fonction, nous évitons des déchargements/rechargements de contexte, les entrées dans le programme, etc., ce qui provoque un meilleur rendement en instructions exécutées de la fonction.

Le phénomène déjà évoqué au paragraphe 4 est particulièrement évident sur ces courbes. Le minimum pour un FDT est la création du contexte nécessaire (produit ou consommé) et sa fermeture (envoi d'un message de fin pour le producteur et prise en compte par le consommateur).

Pour un FDT produit, on observe que le coût initial d'entrée dans le programme est élevé 37 500 instructions, puis que le coût pour chaque FDT est de 25 250 instructions. Si l'on découpe les 3700 instructions (14 %) nécessaires à l'envoi du message de fermeture, on constate que le coût d'une déclaration de FDT représente à peu près la moitié du coût de création d'une AL vide.

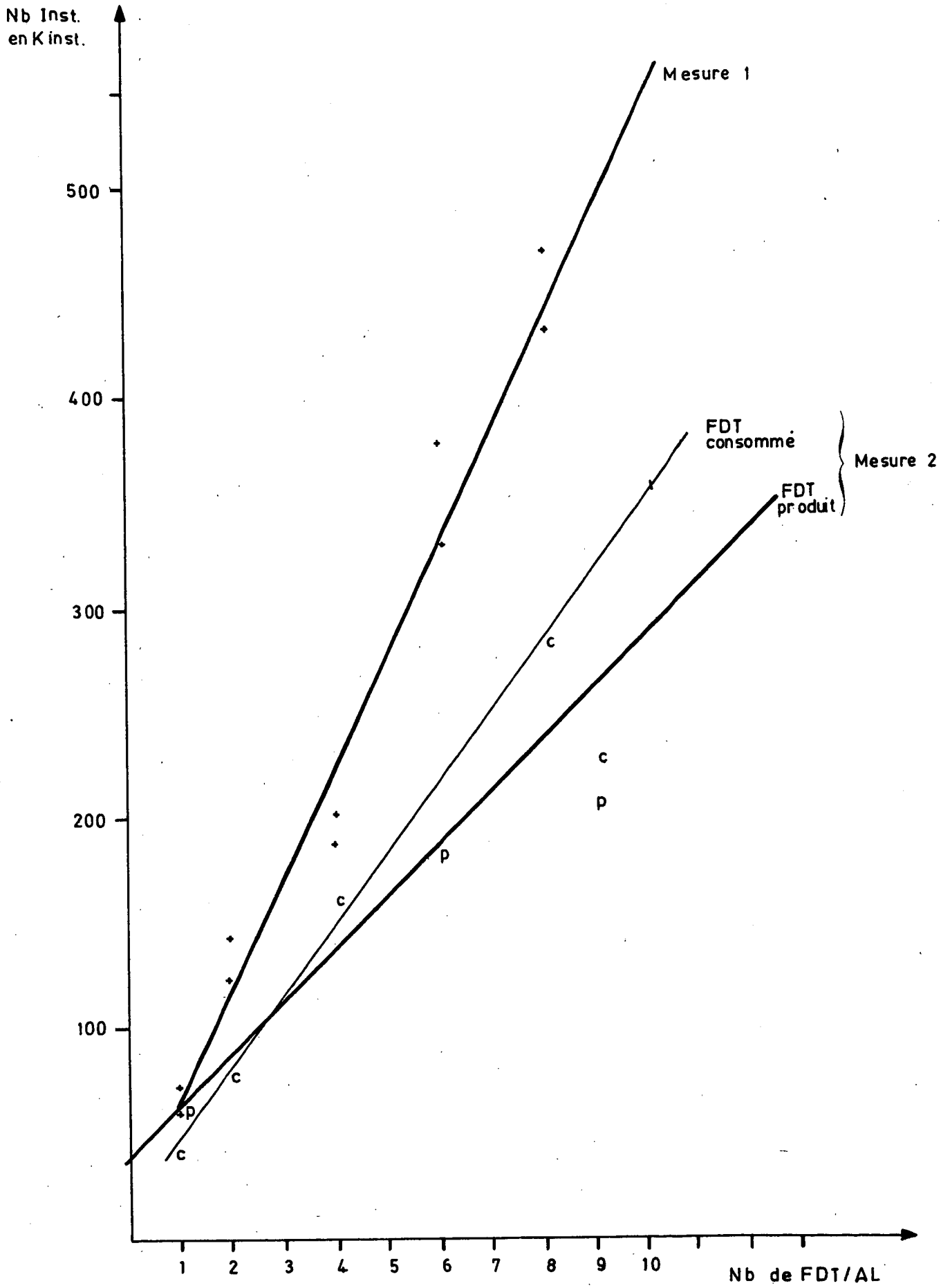


Figure 13 - Coûts d'installation FDT sur les sites locaux .

Coût n FDT = n × 54 000 instructions + 15 000 instructions
Coût n FDT.P = n × 25 250 instructions + 37 500 instructions
Coût n FDT.C = n × 35 000 instructions + 10 000 instructions

Pour un FDT consommé, le coût d'entrée dans le programme semble plus faible par contre, le coût propre à chaque FDT est nettement plus élevé : 35 000 instructions.

Le coût de prise en compte du message annonçant la fermeture est assez faible : 3900 instructions, soit 11 % du coût total. Le coût vient donc principalement de la mise en place du contexte et de la réservation des ressources nécessaires.

Le tableau 3 laisse apparaître que la charge des sites locaux devient suffisamment importante vers 6 à 7 FDT pour que la durée de l'AL soit supérieure à la rotation du jeton. Ce qui explique que le temps de réponse fasse un bond important à 8 FDT. En effet, le processus de soumission des transactions étant cyclique si la fin de transaction se situe juste après le passage du jeton, il faut attendre 18 s avant le prochain passage.

La déclaration des FDT se faisant pendant la phase initiale de lancement des AL, celle-ci se trouve allongée. Par rapport à la figure 7, la figure 14 montre les temps écoulés pour chacune des deux étapes. La phase finale est allongée par l'attente du premier et dernier article du FDT et son interprétation.

6.4. - Coût d'un transfert d'article de FDT

Pour mesurer le coût d'un transfert d'articles de FDT, nous avons repris la mesure 2 précédente avec une AL produisant n articles dans un seul FDT sur le site 1 et une AL consommant tous les articles dudit FDT sur le site 2.

Le mécanisme de transfert d'un article de FDT est décrit sur la figure 13 bis. Il inclut un protocole de contrôle de flux de bout en bout entre les deux processus Producteur FDT et Consommateur FDT.

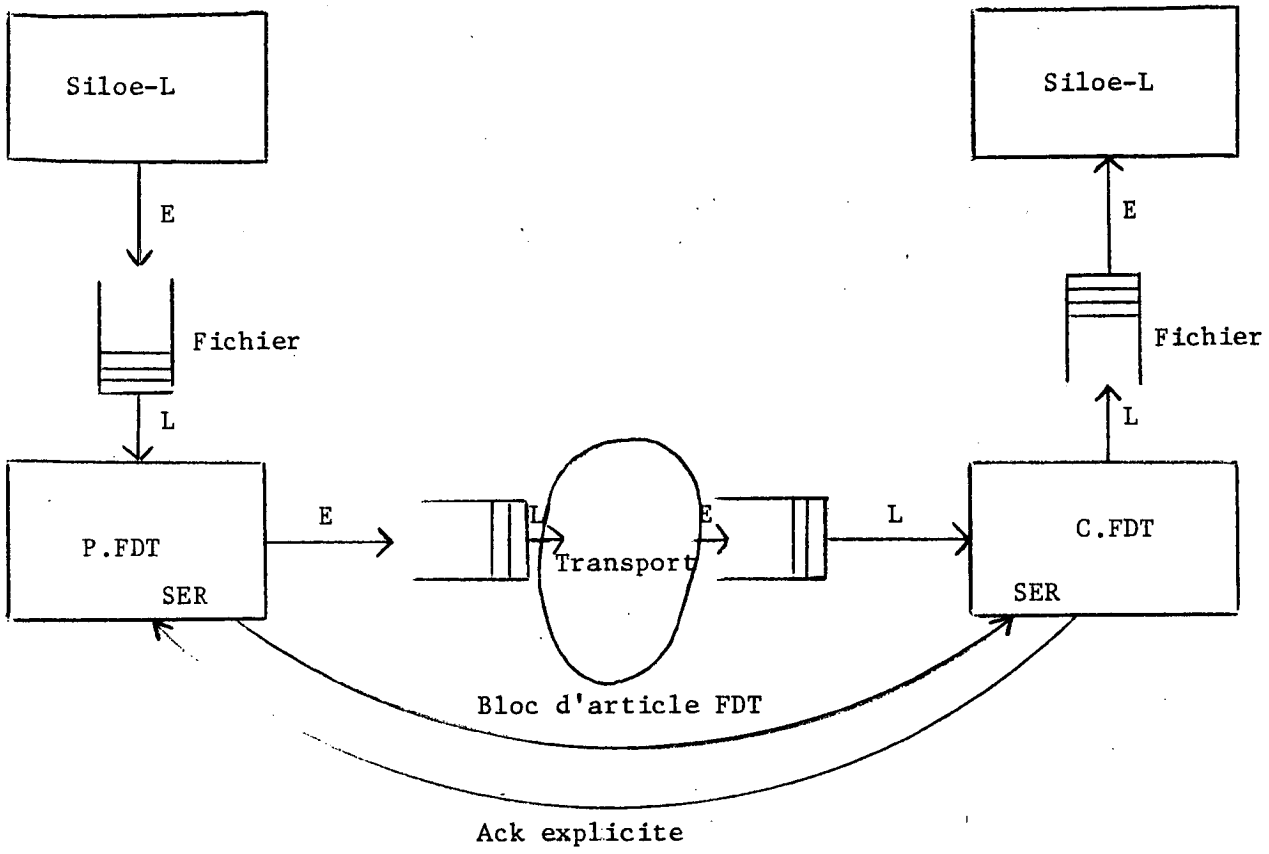


Figure 13 bis

La première implémentation que nous avons mesurée transfère les articles de FDT dès qu'ils sont produits un par un. La figure 15 donne les courbes obtenues. Nous obtenons pour le producteur : N * 18 500 instructions + 95 000 instructions. La constante est égale à 45 000 instructions pour installer l'AL plus 50 000 instructions, soit à peu près le coût de déclaration du FDT vu au paragraphe précédent.

PEX doté de 2 AL n FDT	Site Global	Sites locaux						
		1 FDT produit			2 FDT consommateur			
	Temps de réponse en s	Activité CPU %	Activité CPU %	Activité ligne Global <-> 1 %	Activité CPU %	Activité ligne Global <-> 2 %	Les 3 CPU actifs %	Activité ligne Global 1 <-> 2 %
0 (ligne 2 du tableau 1)	19,1	58	29	5,3	32	5,4	11,6	0,8
1	19,7	57	39,8	5	37,1	5	24,0	2,3
2	19,6	57	45,5	4,9	44,5	5,1	17,5	3,4
4	21,5	56,7	52,7	4,9	56,3	5,1	20,8	5,3
6	21,8	55	60,1	4,8	67	4,8	21,6	7,2
8	34,9	41,5	51,7	3,4	56,2	3,4	15,1	5,9
9	30,4	39,5	51,4	3,2	56,4	3,2	14,5	5,9

Tableau 3 - Activité comparée du site global et des sites locaux avec des AL déclarant des FDT

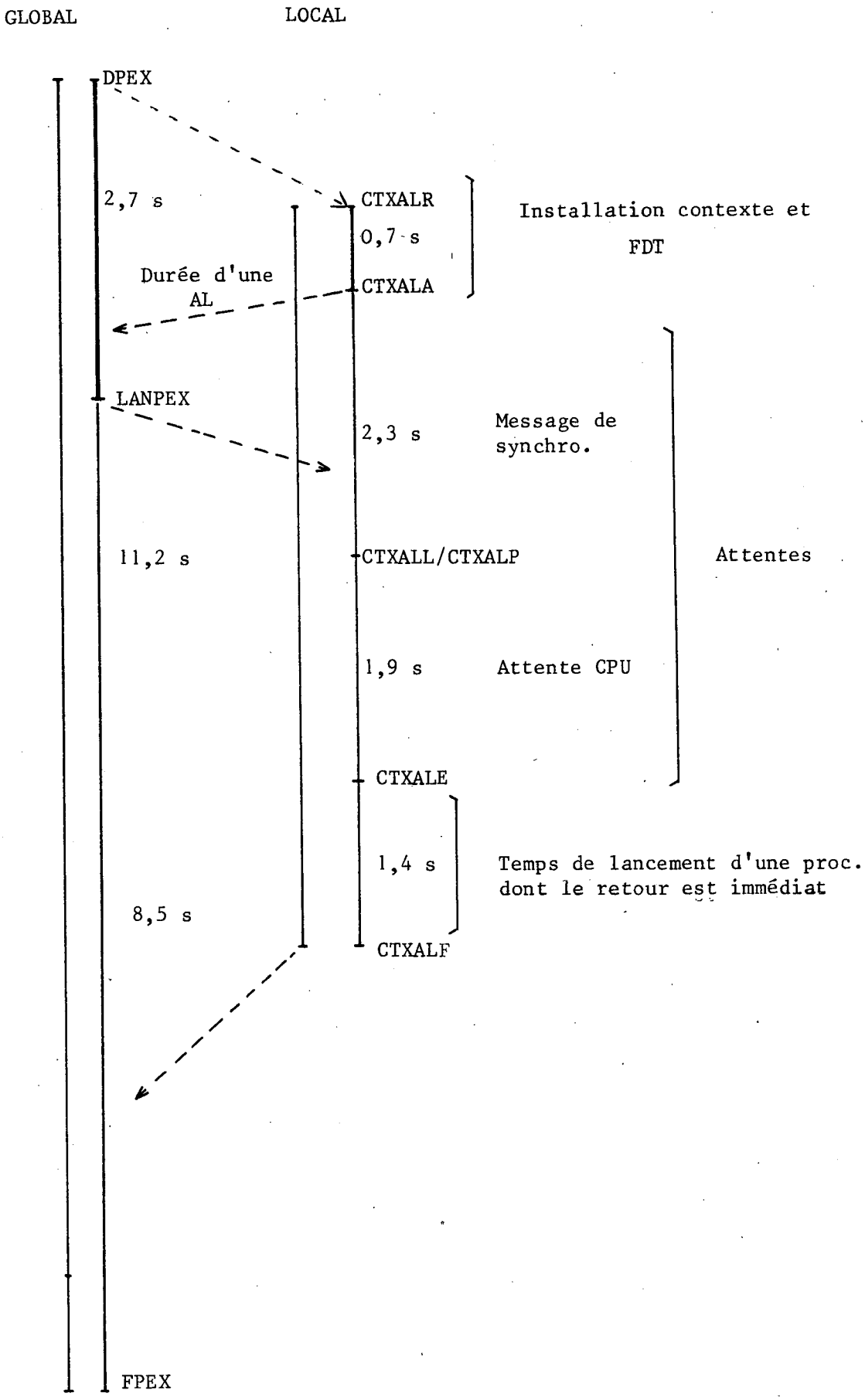


Figure 14 - PEX avec FDT - 3 AL en //. Les 3 sites exécutent la même chose . Période de soumission des PEX : 38 s

La précision de la méthode de mesure ne permet pas de retrouver la valeur donnée par l'équation du paragraphe précédent (62 750 instructions). Ceci étant encore une fois dû au phénomène d'empilement des appels et des couches. Le coût intrinsèque d'un article transféré par le site producteur est 18 500. Le coût d'émission et réception d'un message étant de 3700 + 3900 instructions, soit seulement 41 %, on peut s'étonner du très faible rendement. En particulier si on le compare au coût d'activation des VS. Ceci est dû à une implémentation en BASIC peu efficace. L'article est, en effet, transféré de couche en couche à travers des fichiers $SILOE^{E}\rightarrow{}^{L}SER$, $SER^{E}\rightarrow{}^{L}Réseaux^{L}\rightarrow Transfert$ et l'on a donc 3 $lectures^{L}$ et 2 $écritures^{E}$ fichiers à opérer. Il est évident qu'une implémentation utilisant des pointeurs aurait déjà été plus efficace.

Pour le site consommateur, nous obtenons N * 25 850 + 68 000 instructions. La constante provient des 45 000 instructions de l'AL plus 23 000 instructionspour la création du FDT consommé. Encore une fois, nous ne retrouvons pas exactement les 45 000 instructions attendues suivant les résultats du paragraphe précédent ; en partie à cause du trop faible nombre de points de mesure obtenus. Par article de FDT, nous obtenons 25 850 instructions dont seulement 7660, soit 26 % sont dues à l'émission et réception du message. Le très faible rendement s'explique de la même façon que précédemment par les passages dans les fichiers $^{L}Réseau^{E}\ ^{L}SER^{E}\ ^{L}SILOE$ qui imposent 3 $lectures^{L}$ et deux $écritures^{E}$.

Le très faible rendement de notre implémentation nous a donc très vite conduit à introduire des paliatifs. Nous avons donc groupé les articles produits avant de les envoyer. Le choix du facteur de blocage est bien sûr délicat. En outre, il introduit un retard dans le transfert des articles et donc une diminution de l'utilisation du parallélisme.

Nous avons fait les mesures en utilisant 2 facteurs de blocagesdifférents, 7 par 7 et 10 par 10.

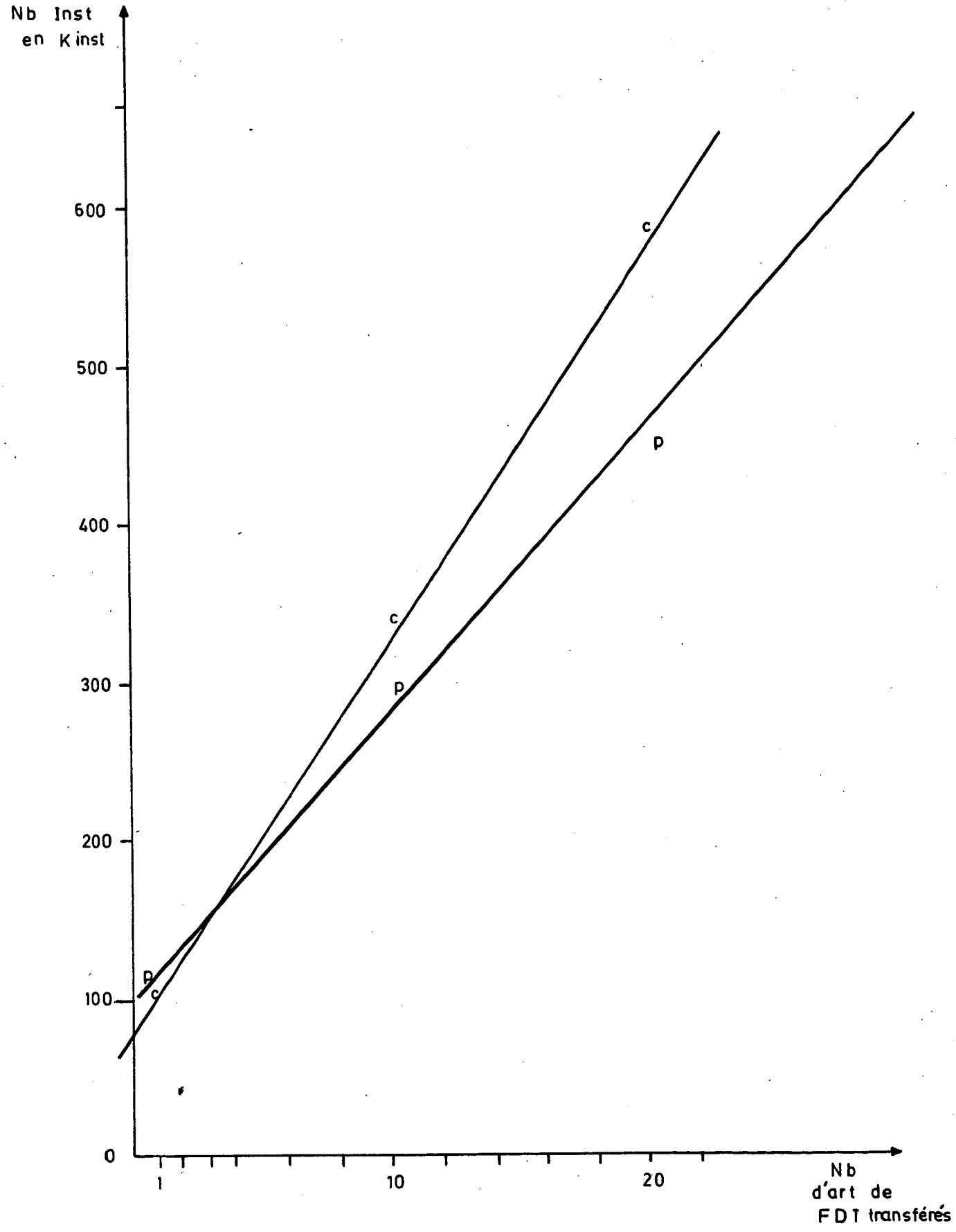


Figure 15 - Coût des articles de FDT transférés 1 par 1

N * FDT.P = N * 18 500 instructions + 95 000 instructions

N * FDT.C = N * 25 850 instructions + 68 000 instructions

Article / article

La figure 16 donne les résultats avec le facteur de blocage 7 par 7, utilisé régulièrement dans les mesures ultérieures. La première remarque est qu'il y a peu de différence entre le coût sur le site producteur et le coût sur le site consommateur. La deuxième remarque est que nous n'avons plus affaire à une droite.

Le tableau 4 montre que l'amélioration obtenue en nombre d'instructions est considérable et que lorsque le facteur de blocage augmente, on tend vers un asymptote très rapidement. La courbe avec un facteur de blocage égal à 10 est très voisine de celle avec un facteur de blocage égal à 7. Par contre, le temps de réponse passe par un minima puis augmente pour de petites quantités d'articles transférés. Ceci s'explique bien évidemment par le fait que l'AL consommatrice commence de plus en plus tard après l'AL productrice et donc nous perdons du parallélisme. Lorsque le nombre d'articles transférés augmente, le temps de réponse converge à nouveau. Il y a donc une courbe de minima de temps de réponse fonction du facteur de blocage et du nombre moyen d'articles à transférer. Il serait intéressant d'utiliser ces mesures pour valider un modèle dans le but de trouver cette loi de minima.

Facteur de blocage \ Nbre d'articles transférés		1	10	20	40	50	90
1	P Kinst	65	291	411			
	Temps de réponse	19,7	37,6	50,3			
	C Kinst	55	302	539			
7	P Kinst		101	149	271	312	615
	Temps de réponse		17,3	19,4	37,9	38,9	61,2
	C Kinst		93	149	270	317	643
10	P Kinst						
	Temps de réponse		17,8	20,1	43	38,3	61,4
	C Kinst		86	141	140	301	568

Tableau 4 - Coût des transferts d'articles de FDT en fonction du facteur de blocage

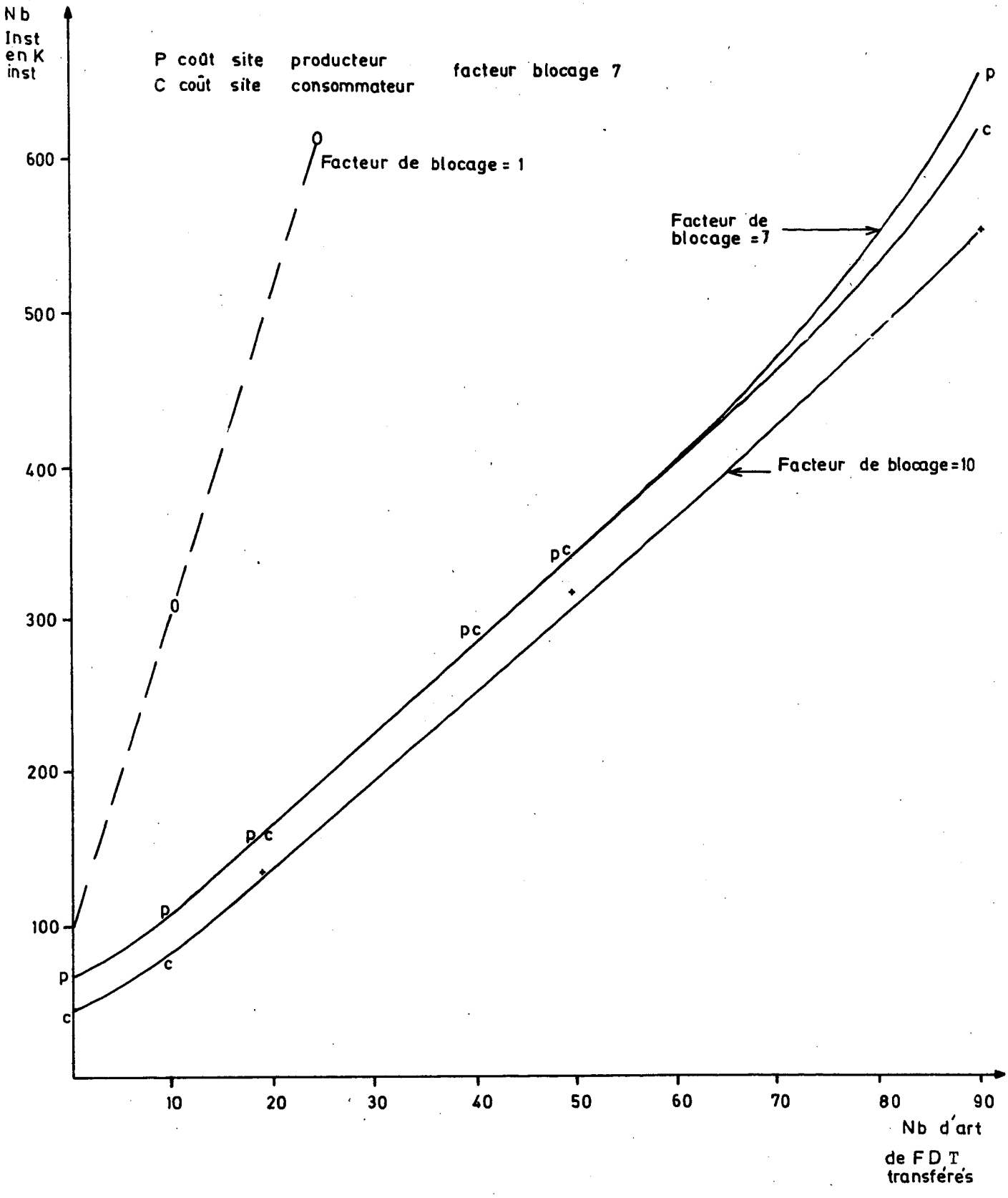


Figure 16 - Coût des transferts d'articles avec facteur de blocage

La dernière mesure que nous avons effectuée consiste à avoir,sur un même site, une AL productrice d'un FDT1 et une AL consommatrice d'un FDT2. Le FDT1 est consommé sur le site 2 par une AL qui produit le FDT2. Ce scénario correspond, en fait, à la production et consommation d'articles pour les mesures sur SILOE (voir § 8). L'intérêt de cette mesure vient du fait que nous avons déjà cité,l'activation de la même fonction simultanément évite, l'exécution d'instructions de sauvegarde de contexte. D'autre part, il est fréquent dans les scénarios BDR qu'une AL consomme un FDT ou utilise des données locales et, en retour, produise un FDT. Nous estimons que notre scénario en est assez voisin et nous utiliserons ultérieurement ces résultats pour évaluer les coûts propres à SILOE.

La courbe de la figure 17 montre que l'allure générale de la courbe n'est pas modifiée. Par contre, nous ne trouvons pas la somme des deux courbes précédentes et elle s'en éloigne même sensiblement. Cette observation renforcera encore notre volonté d'utiliser la méthode de mesure utilisée en s'approchant au mieux des conditions d'utilisation d'un service par les couches supérieures. Réciproquement, comme nous ne pouvons pas, faute de temps et d'imagination, mesurer toutes les conditions réelles, il faudra être prudent lors de l'utilisation de ces résultats pour des modèles et simulations.

La nature d'un problème distribué exige le transfert de données d'un site à un autre et il s'avère que ce transfert est relativement plus coûteux que les mécanismes de base du système réparti. Ceci est à la fois rassurant et inquiétant. Rassurant au sens où l'utilisateur paye bien pour ce qu'il souhaite faire, c'est-à-dire travailler sur des données distantes. Donc optimiser le nombre de transferts de données est un problème important dans une application BDR et cette optimisation étant délicate, le problème est en soit inquiétant. Rassurant aussi parce qu'il apparaît que le noyau d'un système réparti n'est pas le point noir que d'aucuns craignent.

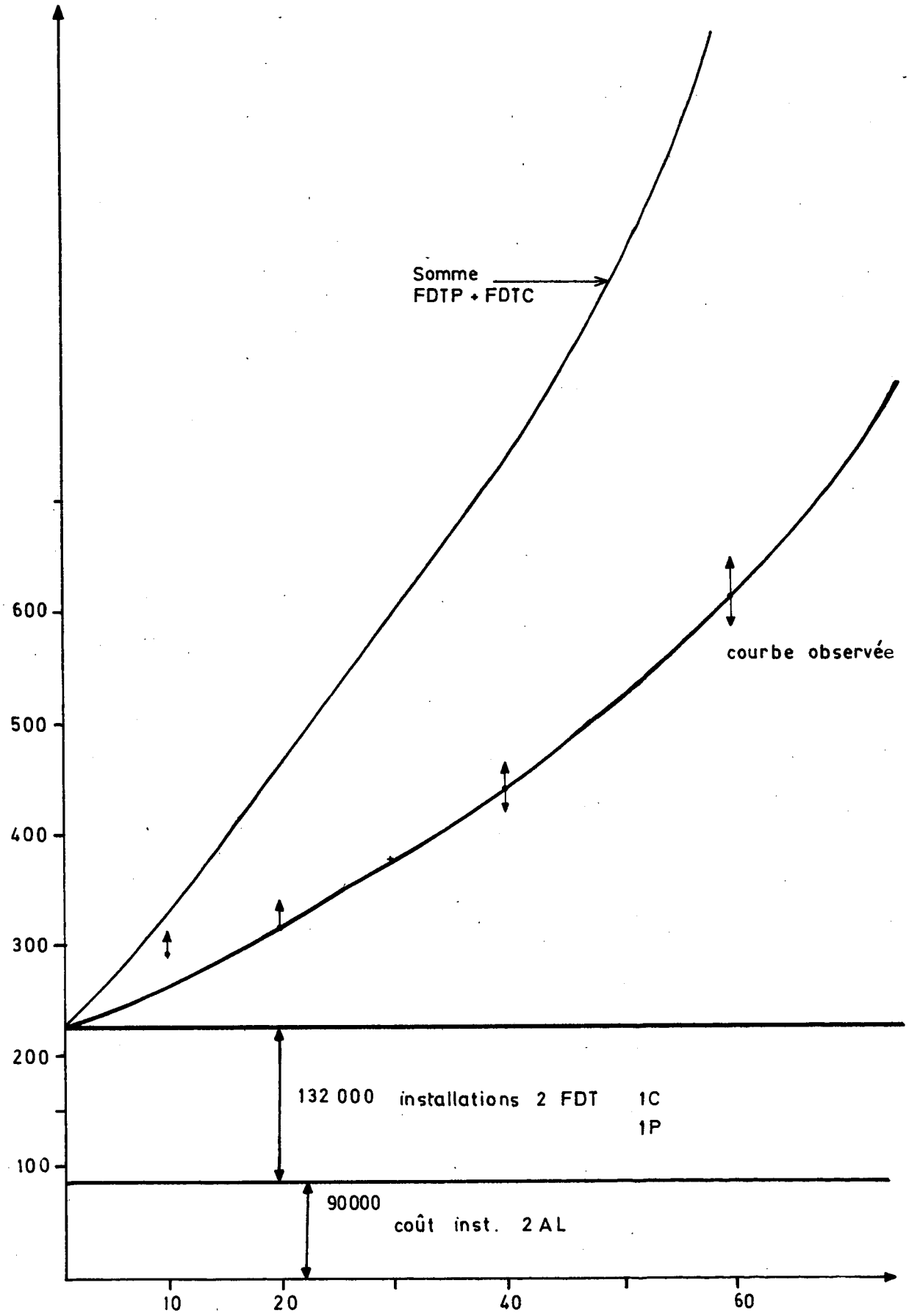


Figure 17 - Coût Production vers 1 site distant et réception d'un FDT distant de N articles FDT sur 1 site unique (facteur de blocage = 7).

6.5. - Conclusion sur SER

Nous avons résumé dans le tableau 4-bis les principaux résultats. La majorité des coûts sont linéaires et proportionnels au nombre d'objets utilisés. Le lecteur remarquera la modestie du coût des variables de synchronisation. Par contre, l'implémentation des FDT est peu efficace. Le transport des articles de FDT est particulièrement coûteux.

	Coût sur le site		
	Global	Local	
Installation de n AL	n * 37 500 + 10 000	29 800 * n + 15 200	
Synchro		Activateur n * 6 000 + 32 000	Récepteur ≃ 6 000 inst. pour 1 VS voir Fig. 10 p. 25
Coût Installation des FDT		Producteur n * 25 250 + 37 500	Consommateur n * 35 000 + 10 000
Transfert d'article FDT par bloc de 7		voir fig. 16 p. 38 ex. 90 000 inst. pour 10 art	

Tableau 14-bis : Récapitulatif SER

7. - SCORE

SCORE a pour fonction de gérer l'accès aux données de manière cohérente. Pour atteindre cet objectif, nous mettons en oeuvre plusieurs principes :

- ordonnancement total des transactions,
- verrouillage décentralisé,
- fichiers temporaires,
- validation à deux phases,
- journaux..

Ordonnancement total des transactions

Tous les producteurs sont installés sur un anneau virtuel [LEL1-79]. Un séquenceur circulant tourne sur cet anneau et permet au producteur qui le possède de prélever des numéros strictement croissants pour les transactions qui lui sont soumises. Ceci nous permet de disposer d'un ensemble de numéros ordonnés pour nos transactions. Un tel numéro est porté par chaque transaction et est appelé estampille. Une description formelle de l'algorithme est donnée dans [ROL 80]. Le coût de l'AV a été étudié précédemment (cf. § 5).

Verrouillage décentralisé

Toute transaction est évaluée et traduite en une suite d'actions locales qui pour s'exécuter ont besoin de ressources. Les ressources prisent en compte par SCORE sont appelées objets et peuvent être modifiables. L'allocation est faite dynamiquement par la transaction avant l'exécution de chaque action locale. Les processus chargés de contrôler l'accès aux objets sont appelés SCORE-Consommateurs (SCORE.C). Chacun d'eux est maître absolu des objets qu'il gère. Plusieurs transactions peuvent s'exécuter en parallèle et les SCORE-Consommateurs traitent les demandes d'allocation les unes après les autres. Il y a donc possibilité d'étreinte mortelle (Dead lock). Pour nous en prévenir, nous utilisons un mécanisme de retour en arrière (Roll Back). Une optimisation de ces retours arrière est obtenue grâce au principe des étapes. La génération systématique de Roll-back étant très difficile, leur coût n'a pu être mesuré.

Fichiers temporaires

Pour pouvoir revenir à une étape précédente sans attenter à la cohérence de la base, il faut être capable de remettre les objets dans l'état qu'ils avaient avant la pose des verrous [SDD1 79]. Pour chaque objet dont l'accès requis est écriture, une copie est faite sur un fichier avant par SILOE local.

Validation à deux phases

Une transaction respecte bien sur le principe du verrouillage à deux phases. Dans un système distribué, il faut que les verrous soient libérés à la fin de la transaction sur chaque site local. En outre, il faut que chaque site local prenne la même décision en fin de transaction et ce même en présence de pannes [STO 79]. Ceci est obtenu grâce au protocole de validation à deux phases.

Journaux

Les aspects,concernant la remise à niveau d'une machine après une panne, nécessitent l'établissement des journaux et de points de reprises. Lorsqu'un site portant un morceau de base dupliqué tombe en panne, il est possible que les transactions concernant ces données continuent à s'exécuter. Le problème consiste à remettre la base de donnée à niveau lorsque la machine en panne revient dans le système. De même, il faut savoir jusqu'à quel niveau on peut faire des copies cohérentes de la base répartie et ce en évitant de verrouiller toute la base [LEL 80]. Il est même souhaitable de pouvoir faire des sauvegardes sur chaque base locale indépendamment les unes des autres tout en sachant définir des points de reprises globaux.

Ce dernier aspect n'a pu être mesuré. Par contre, le coût de maintien des informations sur les journaux est inclus dans les mesures qui suivent.

7.1. - Obtention du ticket

La première tâche d'une transaction consiste à obtenir un ticket. Cette opération est réalisée à l'aide du verbe "Début-Trans". Elle a pour effet sur le site global, en plus de fournir le numéro de ticket, d'intialiser un contexte global pour la transaction. Réciproquement, il faut libérer le contexte en fin de transaction. Ceci se fait au cours de l'exécution du verbe "Fin-Trans". Nous avons donc lancé une transaction cyclique vide réalisant uniquement :

```
Début-Trans
Fin-Trans.
```

Il n'y a donc aucune validation finale le seul travail effectué consiste a :

- obtenir un ticket,
- créer le contexte global de la transaction,
- constater qu'aucun site local n'a été impliqué dans cette transaction et libérer le contexte global.

Le coût global s'avère être de 111 000 instructions. Ce coût élevé ne peut guère s'expliquer que par la création/libération des contextes dans les deux couches traversées SILOE et SCORE.

7.2. - Pose des verrous et validation

Les deux primitives ne sont pas séparables dans la mesure, où les mécanismes de verrouillage à deux phases et de validation à deux phases empêchent leur séparation. Le site global n'est pas logiquement concerné par les verrous posés, en fait il conserve seulement trace des sites atteints,les demandes de verrous étant incluses dans les contextes d'AL. Les mesures que nous avons faites sont hélas très limitées par le prototype, nous ne pouvions poser au maximum que 3 verrous et atteindre au maximum deux sites distants. Ceci limite ici considérablement la validité des mesures qui ne peuvent donner qu'une impression. On relativisera donc les résultats obtenus du fait de leur très faible précision. Ces résultats ont été obtenus en lançant une transaction à 1 PEX dotée de 1 ou 2 AL suivant le nombre de sites à atteindre, chaque AL demandant 1, 2

ou 3 verrous. Sur le site global, on obtient le coût de la validation à deux phases du côté du producteur et sur le site local, nous avons le coût de la pose des verrous, de la validation finale et de la journalisation.

La figure 18 donne les résultats obtenus sur le site global. Comme, nous nous y attendions le nombre de verrous ne semble pas influer sur la validation finale. Seul le nombre de sites joue un rôle. La validation pour 1 site atteint coûte environ 12 000 instructions, pour deux sites, environ 23 000 instructions.Il aurait fallu au moins 4 ou 5 sites pour chercher une courbe néanmoins, nous pouvons constater que ce coût est dû presque exclusivement à la gestion des messages échangés.

Coût validation finale :

			Nombre de sites 1	2
PTC	émis	3700	3700	7400
ACK PT	reçu	3900	3900	7800
Commit	émis	3700	3700	7400
			11300	22600

Il y apparaît ici très nettement que si les messages PTC et Commit pouvaient être diffusésetnon envoyés séparément pour chaque site, le coût sur le site producteur serait largement abaissé.

Le tableau 5 fait, en outre, apparaître que la validation à deux phases ne modifie pratiquement pas le temps de réponse de la transaction. Le temps dû à la période de rotation du séquenceur joue bien sûr mais on remarquera que si les sites locaux sont nettement plus chargés, ils ne sont pas saturés. On peut donc penser que le protocole de validation à deux phases n'allonge que très peu le temps de réponse. On remarquera, en outre, que l'augmentation de la charge des lignes du site global vers les sites locaux, due à la validation à deux phases, est relativement faible, passage de 5,3 % à environ 6,8 %, soit une augmentation de 1,5 % seulement.

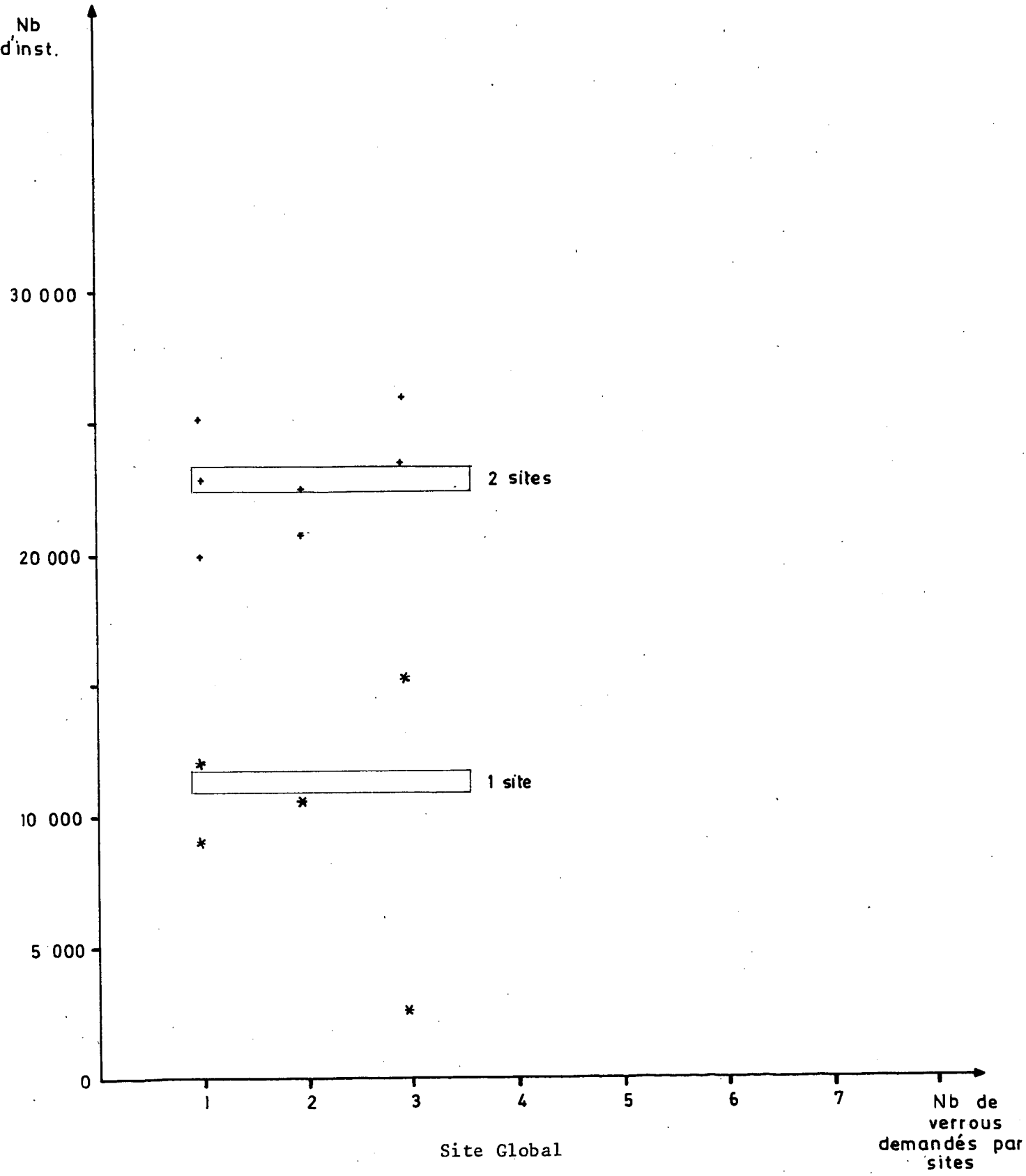


Figure 18 - Mécanisme de validation à 2 phases sur le site global
(le nombre de verrous demandés ne semble pas intervenir)

Trans. dotée de 1 AL / Nb de verrous par AL		Site Global		Sites locaux 1		Sites locaux 2			
		Temps d'exécution de la transaction en s	Activité CPU %	Activité CPU %	Activité ligne Global <-> 1 %	Activité CPU %	Activité ligne Global <-> 2 %	Les 3 CPU actifs %	Sites locaux 1 et 2 actifs simultanément %
0		18	51	33,5	5,3	20	1	3,7	7,0
1	L	19	54,3	52,5	6,9	25,3	1,1	6,7	14
	E	18,9	53,8	54,2	6,9	25,7	1,1	6,6	14,4
2	L	18,8	58,2	60,2	6,9	25,7	1,1	8,3	15,6
	E	19	54	60,4	6,8	25	1,1	7,2	15,3
3	L	19,8	53	66	6,5	20	1,1	6,5	12,7
	E	18,5	53,2	67	6,8	25	1,1	8,5	16,5
2 AL Nb de verrous par AL						pas d'AL			
0		19,1	58	29	5,3	32	5,4	11,6	14,77
1	L	19,7	63,5	51,8	6,7	51,8	6,7	24	41,5
	E	19,4	63	57	6,7	54	6,8	24,2	44
2	L	19,4	63	58,4	6,6	58,4	6,7	24,7	47,4
	E	19,6	62,6	63,6	6,7	60,3	6,8	25,2	51
3	L	19,9	62,5	68	6,6	64,8	6,7	27	54,8
	E	19,8	62,2	68,3	6,5	66,7	6,6	26,8	56,2

Tableau 5 : Activité comparée des machines lors de la validation à deux phases

La figure 19 donne le nombre d'instructions exécutées sur un site local pour poser N verrous,valider la transaction et libérer ces verrous.Malgré une relative dispersion des points, nous pouvons extraire une droite d'équation :

N*(verrous+validation) = N * 30000 instructions + 90000 instructions.

Nous pouvons donc en déduire que la validation à deux phases sur un site local coûte 90000 instructions. Parmi ces 90000 instructions, 11500 sont dues à la gestion des messages du protocole. Le reste est principalement dû au coût de la journalisation. Il y a, en effet, écriture de deux articles sur le journal. Ceci explique la différence de coût entre le site global et le site local. Rappelons que le mécanisme de journalisation n'est pas propre aux systèmes distribués. On remarquera que son coût est largement supérieur à la gestion des messages.

Le coût propre de la pose et libération d'un verrou semble être de 30000 instructions.

Ce coût est extrémement élevé si on le compare, par exemple, à la prise en place et au lancement d'un AL. N'oublions pas qu'à chaque pose de verrou, le mécanisme de prévention des interblocages est activé. Dans cette mesure il n'y a jamais déclenchement de roll-back, néanmoins, la complexité de la gestion des tables peut expliquer à elle seule ce coût.

Sur le tableau 5, nour remarquerons que, bien sûr, le taux d'activité des sites locaux fait un bond dès qu'un premier verrou est posé. Ceci est dû bien sûr au fait que le mécanisme de validation n'est pas activé lorsqu'aucun objet n'est utilisé sur un site déterminé. Le taux d'activité augmente après régulièrement en fonction du nombre de verrous posés. La deuxième remarque importante porte sur l'activité parallèle des sites locaux. Par exemple, pour un PEX à 2 AL et 3 verrous Ecriture,on observe un taux d'activité des sites locaux 1 et 2 de 68,3 % et 66,7 % et un taux d'activité parallèle de 56,2 %, soit environ 80 % du temps. Il y a donc bien ici utilisation du parallélisme au maximum.

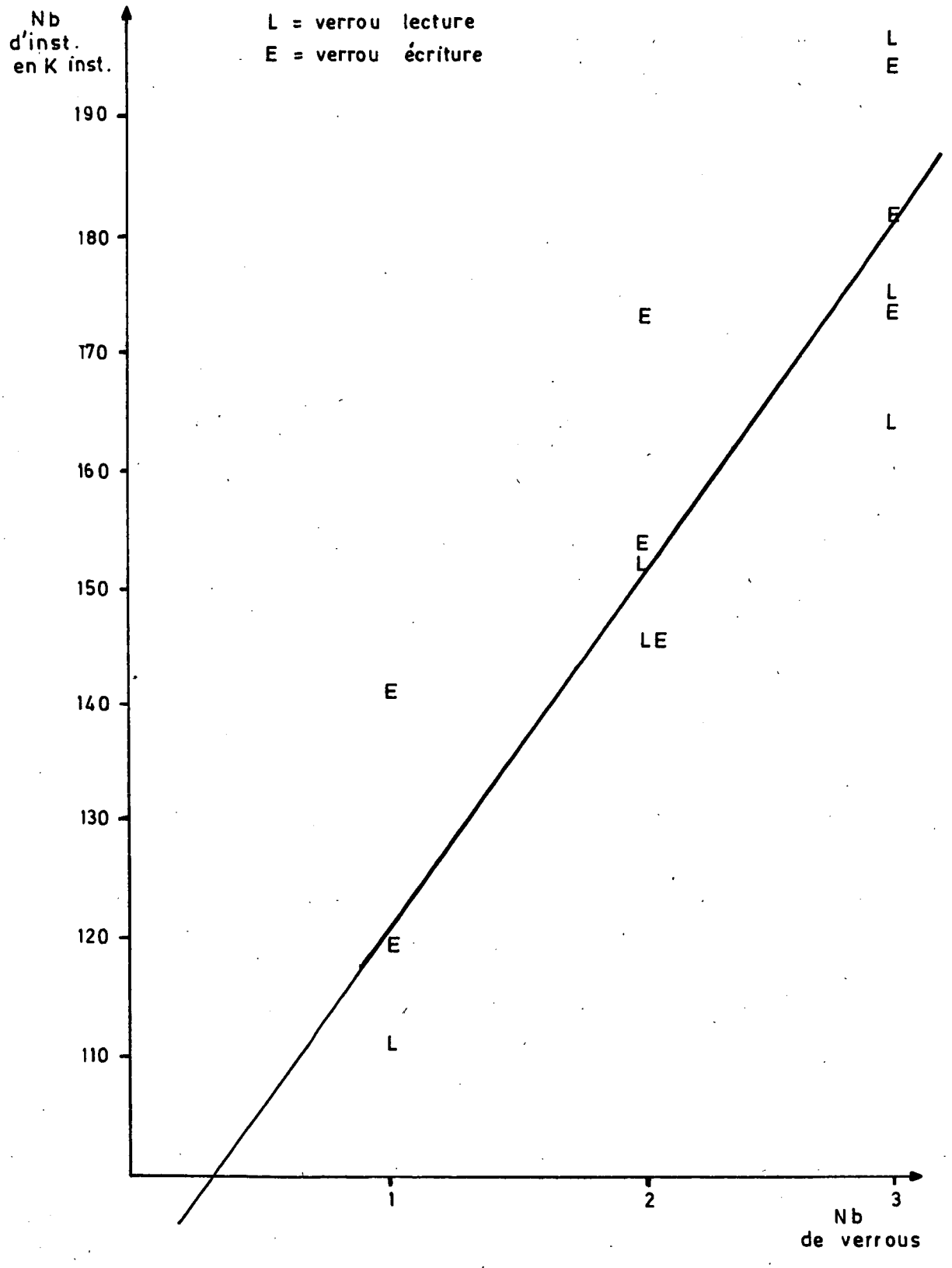


Figure 19 - Sites Locaux - Pose des verrous et validation

Pose de N verrous + validation = 90 000 instructions + N
× 30 000 instructions

Ce phénomène était, d'ailleurs,amusant à voir sur le barygraph du moniteur matériel, un balancement des activités du site global et des sites locaux. Site global très actif => sites locaux peu actifs (attente) et réciproquement. Ceci est moins visible sur le tableau 5 mais on peut, tout de même, remarquer que le taux d'activité simultané des 3 machines est de moitié inférieur au taux d'activité simultané des sites locaux.

7.3. - Prise des images avant

La possibilité de conflits et leursrésolutionsimplique la possibilité d'exécuter des roll-back. Dans ce cas, il faut remettre la base dans l'état où nous l'avons trouvée avant l'accès et la modification de l'objet. Pour des raisons de facilité d'implémentation, SIRIUS-DELTA écrit directement dans la base après avoir conservé une copie de l'objet. Nous appelons ce mécanisme prise d'images avant.

Nous avons donc cherché à évaluer le coût de celui-ci au sein d'une transaction. Pour ce faire, nous avons lancé des transactions à une seule étape ou PEX. Dans ce PEX, une AL opère n écritures sur la base. Ces écritures provoquent à chaque fois la prise d'une image avant. Lors de la validation finale, toutes les images avant sont libérées. La figure 20 donne les résultats obtenus. Une première courbe montre le coût d'une écriture simple sur un fichier R2000. Il s'agit d'une droite montrant que chaque écriture coûte environ 760 instructions. La courbe avec prise d'image avant n'est pas linéaire. Le coût initial d'entrée dans le programme est d'environ 32000 instructions et correspond à l'ouverture du fichier et à la réservation d'espace pour le programme de prise d'image avant. Le coût de la prise et de l'écriture est probablement linéaire. Par contre, lors de la validation finale, il faut libérer toutes les ressources acquises et, entre autre, l'espace alloué aux n images avant. Cette libération est faite en une seule fois mais, bien sûr, son coût est proportionnel au nombre d'images avant prises. C'est ce dernier phénomène qui explique la forme générale de la courbe.

Remarquons que ce coût n'est pas spécifique à un système distribué mais inhérent à tout système transactionnel où l'on veut résoudre les problèmes de dead-lock et de reprise.

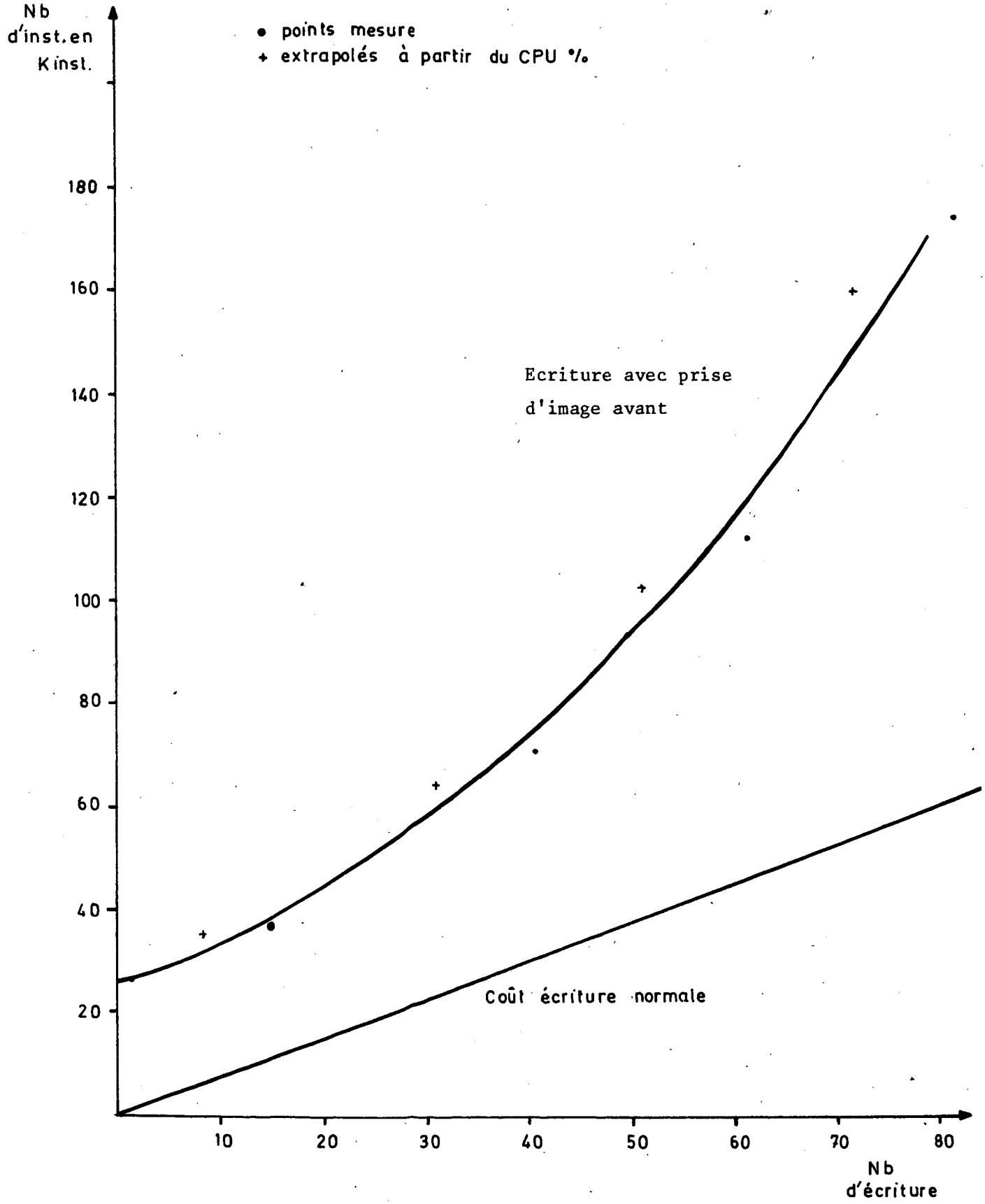


Figure 20 - Coût des images avant

n écritures normales = 758 * n

7.4. - Conclusion

Les coûts de la couche SCORE sont pour une part propres à tout système de base donnée où le partage de la base entre plusieurs usagers est prévu (verrous) et où des reprises après pannes ou erreurs sont possibles (journaux). Ces coûts, on le remarquera sont assez élevés. La partie propre au système distribué est la validation à deux phases. Son coût est assez élevé sur chaque site local. Sur le site local, il comprend la libération des images avant. La prise de ticket sur le site global apparaît par contre particulièrement coûteuse, mais n'est exécutée qu'une seule fois par transaction.

	Global	Local
Prise du ticket et Contexte global	110 000	
Validation à 2 phases	n * 12 000	90 000 inst. par site validation à 2 phases + journaux
Verrous , pose et libération		n * 30 000
Prise , Image avant pour les RB		voir figure 20 p. 51 37 000 inst. pour 10 objets

Tableau 5-bis : Récapitulatif des coûts de SCORE

8. - SILOE

Au niveau global, le mécanisme de Décomposition de Requête est assuré par SILOE. C'est SILOE qui identifie la stratégie à mettre en oeuvre pour une requête.

Aussi, le traitement d'une requête globale se fait en plusieurs étapes :

1) Transformation globale-locale, c'est-à-dire découpage de la requête en un ensemble de sous-requêtes ne faisant intervenir qu'un seul site. A ce niveau la connaissance des critères de répartition et des caractéristiques de la duplication des données peut permettre d'éliminer les sites ne contenant pas de données requises.

2) Evaluations à priori des scénarios d'exécution possibles. Il faut trouver une meilleure synchronisation, sinon la meilleure, parmi celles possibles, compte-tenu de certains paramètres (transfert minimum des données le plus souvent mais aussi temps de réponse ou encore temps CPU, parallélisme maximal, répartition des charges ...). Interviennent également des facteurs dynamico-statiques tels que l'existence de copies de données locales ou dynamiques telles que l'état du réseau.

3) Génération du PEX : celui-ci représente le scénario sélectionné, les sous-requêtes locales étant entièrement localisées sur le réseau.

Au niveau local ,c'est SILOE qui soumet la sous-requête local au SGBD et qui, en cas de requête modifiant la Base, saisit l'"image-avant" de l'objet qui va être modifié afin d'être en mesure d'exécuter une demande de roll-back formuléepar SCORE (cf. § 7).

Les différents mécanismes de traduction (global/langage-pivot, langage-pivot/local) sont à la charge de SILOE.

La transformation globale-locale de la requête consiste à identifier les opérations élémentaires (à savoir restriction, projection, jointure, union, ...), à effectuer sur les données locales compte-tenu :

- de la requête gobale,
- des règles de répartition stockées dans le Schéma Interne Global.

Dans son principe, l'opération est relativement simple :

1) traduire la requête globale en algèbre relationnelle pour faire apparaître les opérations élémentaires à appliquer sur les relations globales de la BDR.

2) remplacer la référence à chaque relation globale de la BDR par l'enchaînement correspondant des opérations élémentaires portant sur les données locales.

Le lecteur intéressé trouvera dans [GLOR 80], [STAN 81] tous les éléments nécessaire à la compréhension de SILOE.

Nous voyons donc qu'une requête est d'abord analysée, décomposée et soumise sous forme de langage relationel dans un PEX. La partie analyse, décomposition et production est centralisée sur le site global. En fait, cette partie est dépendante de la complexité de la requête. Nous n'avons pas réussi à discriminer des éléments constitutifs simples caractérisant la complexité d'une requête et, de ce fait, nous n'avons pas pu faire de mesures sur le site global. Par contre, sur le site local, le coût d'une requête peut être exprimé par le nombre de fois où les opérateurs relationels élémentaires sont activés. C'est donc sur le coût d'exécution de ces opérateurs : restriction, projection, jointure et union que nous avons fait porter notre effort. La nature de ces opérateurs n'ayant rien à voir avec un système distribué, nous avons le choix entre mesurer le coût de ces opérateurs activés sur des entrées/sorties locales ou distantes. Pour des raisons de facilité de mise en oeuvre, nous avons préféré opérer sur des données distantes FDT. L'inconvénient en étant bien sûr que le coût des opérateurs est plus ou moins noyé dans le bruit de fond dû aux transferts d'articles de FDT.

8.1. - Coût de l'union

L'union est une opération qui prend deux relations en entrée et en fournit une seule en sortie. La figure 21 montre comment a été effectuée la mesure. On remarquera la symétrie dans les FDT ouvert et fermé sur les sites et dans le nombre d'articles transférés. Cette symétrie a été utilisée pour obtenir les résultats de la figure 22. En effet, le site 2 fait pratiquement la même chose que le site 1 avec ses FDT, par contre, il réalise l'opération d'union en plus. Nous avons ici négligé la différence observée au § 6 entre la création d'un FDT produit et consommé. Le coût de création et mise en oeuvre d'un nombre différent d'AL sur chaque site a, par contre, été pris en compte.

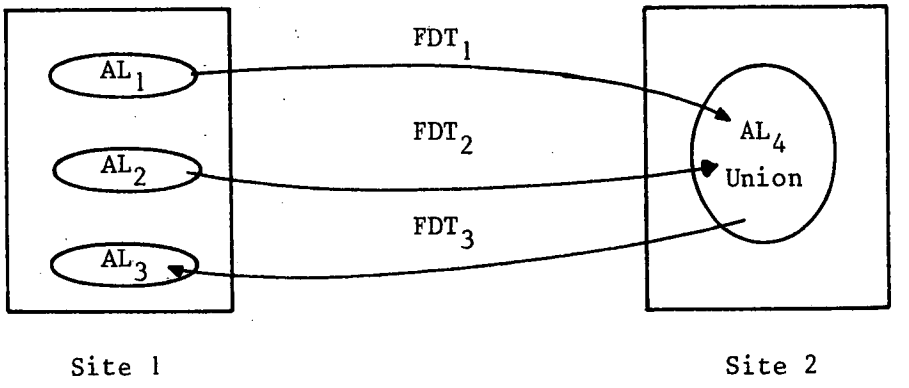


Figure 21 - Schéma de mesure de l'union

Sur la figure 22, on observe que le coût de chaque exécution de l'opérateur union est d'environ 700 instructions. Ce coût est évidemment très faible comparé au coût des transferts d'articles de FDT. Le coût initial semble assez élevé : 91000 instructions. Il est, principalement, dû à l'interprétation de la sous-requête locale reçue par l'AL.

Le tableau 6 fait apparaître que les deux sites travaillent totalement en parallèle, l'attente de l'une par rapport à l'autre est négligeable. La différence porte sur la production par le site 1 du premier article de FDT (en fait des 7 premiers articles) attendu par le site 2 et à la fin, le temps pris par l'AL3 du site 1 pour consommer les derniers articles du FDT.

Le lecteur gardera à l'esprit le coût très élevé des transferts d'articles de FDT. Dans le cas présent, ce sont eux qui saturent les CPU et non l'opération d'union proprement dite.

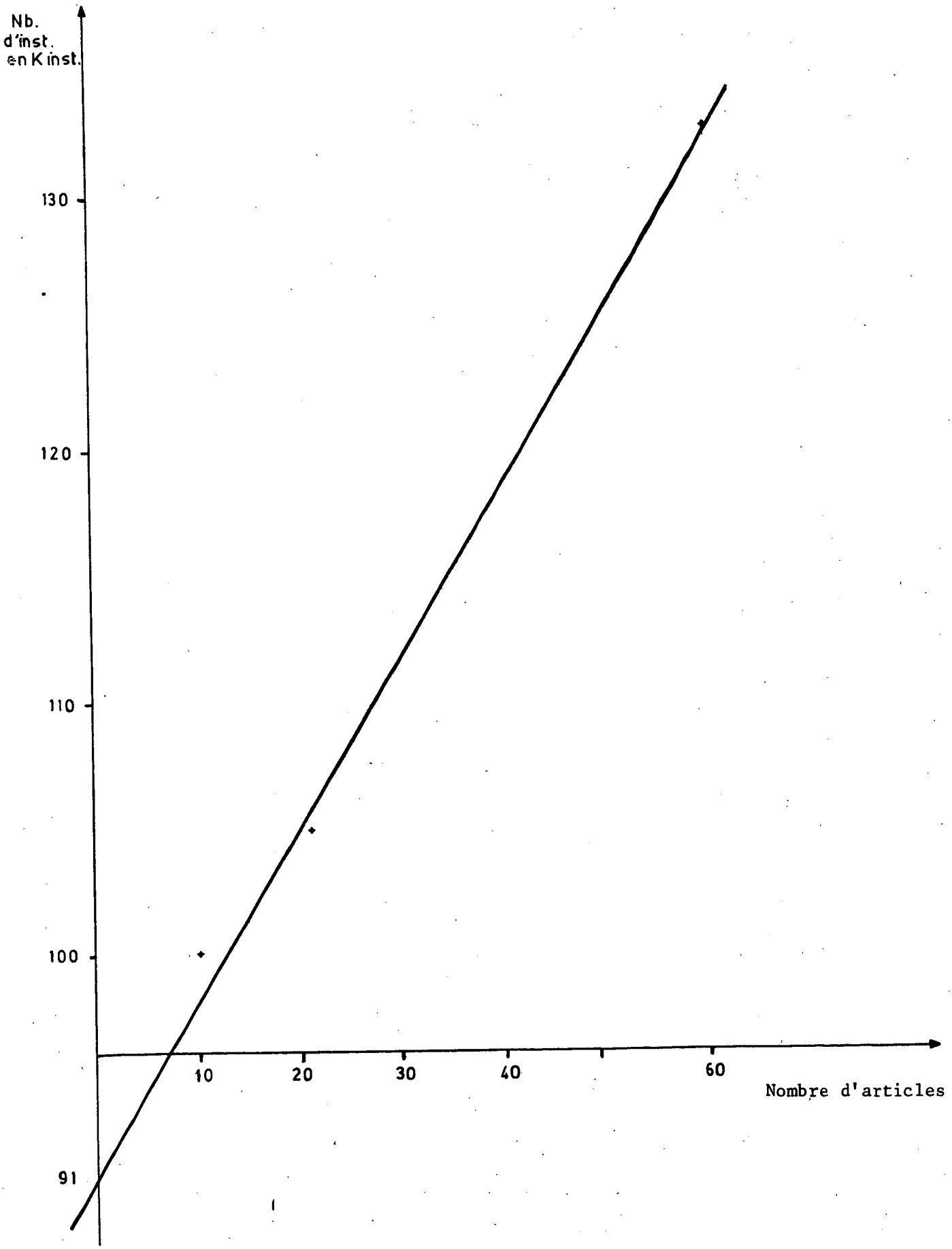


Figure 22 - Coût de l'UNION

n art union = 91 000 + 700 * n

Nb d'union	Site Global	Site local 1		Site local 2			
		3 AL		1 AL où est faite l'union			
	Temps de réponse en s	Activité CPU %	Global<-> M_1 %	Activité CPU %	Global<-> M_2 %	M_1 <-> M_2 %	Activité des 2 CPU locaux en %
10	41,3	60,7	6,4	61,7	3	4,8	49,6
20	41,3	72,3	6,3	74	2,9	6	64,9
40	60,4						
60	80,8	75,9	3,6	79,1	2	7,2	70,2

Tableau 6 - Activité pendant l'union

8.2. - Jointure

La jointure de deux relations réussit lorsque chacune des deux relations possède une même propriété, le résultat est une nouvelle relation. Par contre, s'il y a échec, aucune relation n'est produite. Nous avons donc fait les deux mesures. Le principe de mise en oeuvre est le même que celui décrit sur la figure 21 en remplaçant l'AL du site 2 par une AL demandant une jointure.

Nous présentons la figure 23. afin de bien mettre en évidence le coût déterminant des transferts d'articles de FDT dans le coût global. On retrouve bien évidemment la même allure de courbe qu'au § 6.4. La figure 24 reprend le coût de jointures après avoir déduit le coût des FDT. Il y apparaît un coût initial d'environ 30000 instructions. Le coût d'une jointure réussie est de 1560 instructions et le coût d'une jointure infructueuse est de 940 instructions. Encore une fois, nous soulignerons combien ces coûts sont modérés, comparés aux transferts d'articles de FDT.

Le tableau 7 fait ressortir le taux de parallélisme élevé atteint entre les deux sites locaux. On remarquera que les temps de réponse pour un même nombre d'opérations sont beaucoup plus élevés que dans le cas d'une union. Cette différence est ici principalement due à la différence de coût entre ces opérations.

8.3. - Projections

La projection consiste à éliminer d'une relation un certain nombre d'attributs afin de générer une nouvelle relation. Il y a donc lieu de mesurer le coût de l'élimination de n attributs dans la relation initiale. Les résultats semblent très difficiles à exploiter. En effet, la figure 26 donne les valeurs brutes obtenues sur le site exécutant la projection. Il apparaît, au premier abord, que les points sont à peu près alignés et que le coût n'est pas significativement différent suivant le nombre d'attributs réduits. Le fait que les points soient alignés est assez étonnant, nous devrions trouver la forme de courbe due aux transferts de FDT.

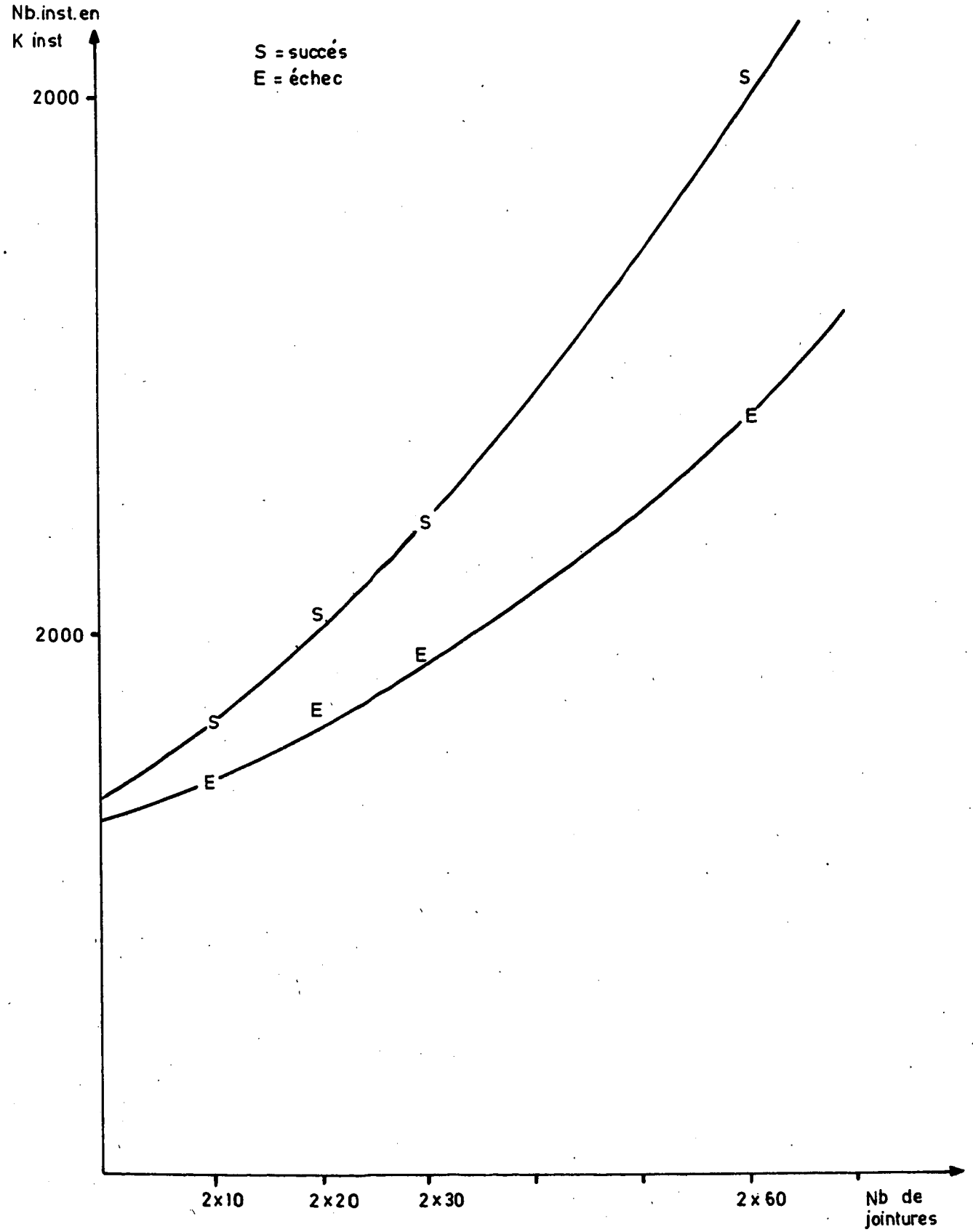


Figure 23 - Jointures

Total transferts inclus

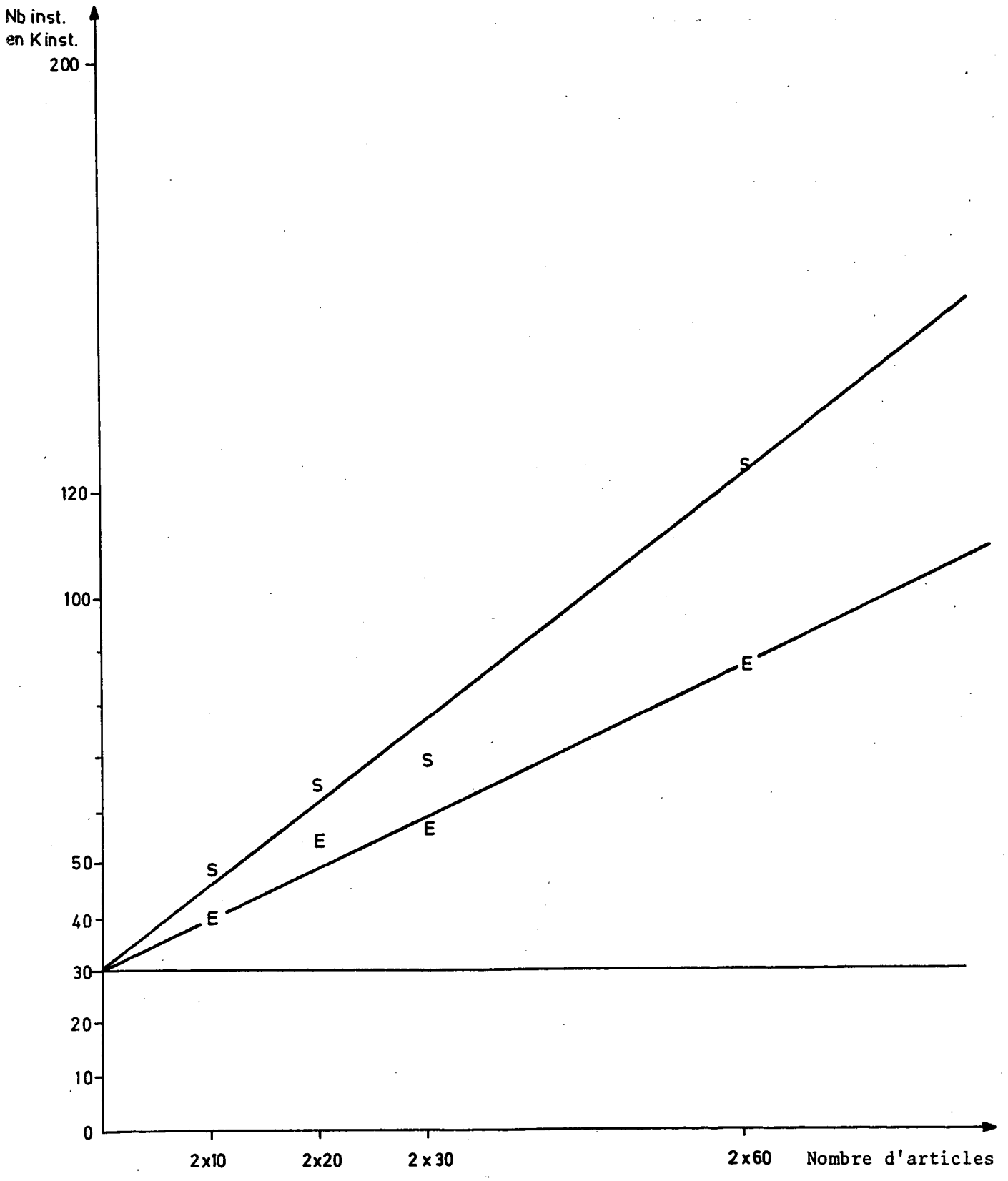


Figure 24 - Coût des jointures

n Succès = n * 1 560 instructions + 30 000 instructions
n Echecs = n * 940 instructions + 30 000 instructions

Nb de jointures		Site Global	Sites locaux					
			1 3 AL		2 1 AL de jointure			
		Temps de réponse en s	Activité CPU %	Activité CPU %	Global<->1 %	Activité CPU %	Global<->2 %	Global 1<->2 %	CPU 1 et 2 actifs %
10	S	67,2	34,5	47,6	4,3	68,5	2,2	3,2	37,8
	E	57,8	37	47,6	4,8	69	2,4	3,2	37,7
20	S	77,6	32,8	43,9	3,9	71,4	2	3,5	36,3
	E	65,3	35	44,2	4,4	70,4	2,3	3,3	36,5
30	S	94,6	29,5	45,9	3,3	70,1	1,8	3,7	38
	E	76,2	31,9	44,8	3,8	67,7	2	3,6	35,8
60	S	135,3	25,1	45,5	2,4	78	1,4	3,6	39,5
	E	98,4	32,3	48,2	3	74,4	1,6	3,7	39,2

Tableau 7 - Activités comparées lors d'une jointure

La figure 27 donne les points obtenus en retirant le nombre d'instructions dues au transfert de FDT. La première remarque est l'extrême dispersion des mesures. La seconde consiste à remarquer que,de toute façon, le coût initial d'entrée dans le programme est très élevé,environ 100 000 instructions. En outre, si l'on se fie au point obtenu pour 1 attribut réduit, le coût par article est assez élevé.

Revenons donc sur chaque aspect : l'initialisation est effectivement coûteuse puisqu'il faut traduire la phrase en langage relationnel pivot en"français"langage local de chaque R2000 . Cette phase d'initialisation est d'autant plus coûteuse que le nombre d'attributs réduits est faible comparé au nombre total d'attributs. Ceci apparaît clairement sur la figure 28 où nous avons porté, pour un même nombre de relations, des projections différentes produisant une nouvelle relation après réduction de 1 à 5 attributs. Le seul coût qui peut varier ici est bien sûr le coût d'initialisation. Il apparaît que l'interprétation de chaque attribut non réduit coûte en plus 6000 instructions. Nous n'avons pas eu la possibilité de faire varier le nombre d'attributs de la relation, ce qui nous interdit de généraliser.

Le coût de la projection d'une relation semble être indépendant du nombre d'attributs réduits. En utilisant la figure 27 et avec toutes les réserves dues à l'imprécision de la mesure, on peut estimer que la projection sur une relation coûte environ 2000 instructions.

8.4. - Restrictions, sélection

La restriction est une opération qui sélectionne les seules relations vérifiant une condition ou prédicat. Il y a donc une relation en entrée et la même ou aucune en sortie. Cette opération est équivalente au verbe Select de français, nous n'avons pas cherché à le mesurer. Seul le coût d'initialisation a pu être évalué et est de l'ordre de 120 000 instructions.

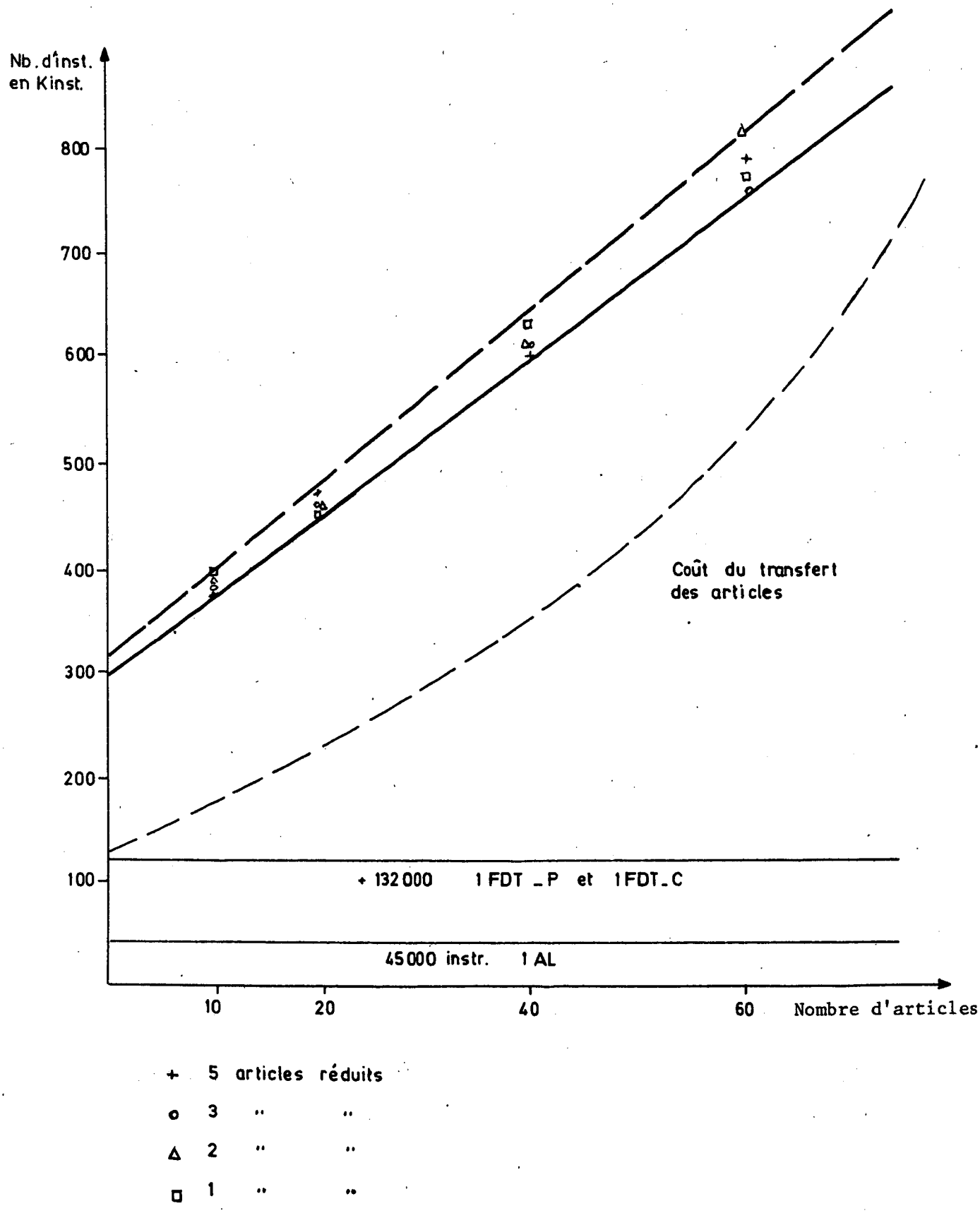


Figure 26 - Projections

On ne retrouve pas l'hyperbole des transferts d'articles.

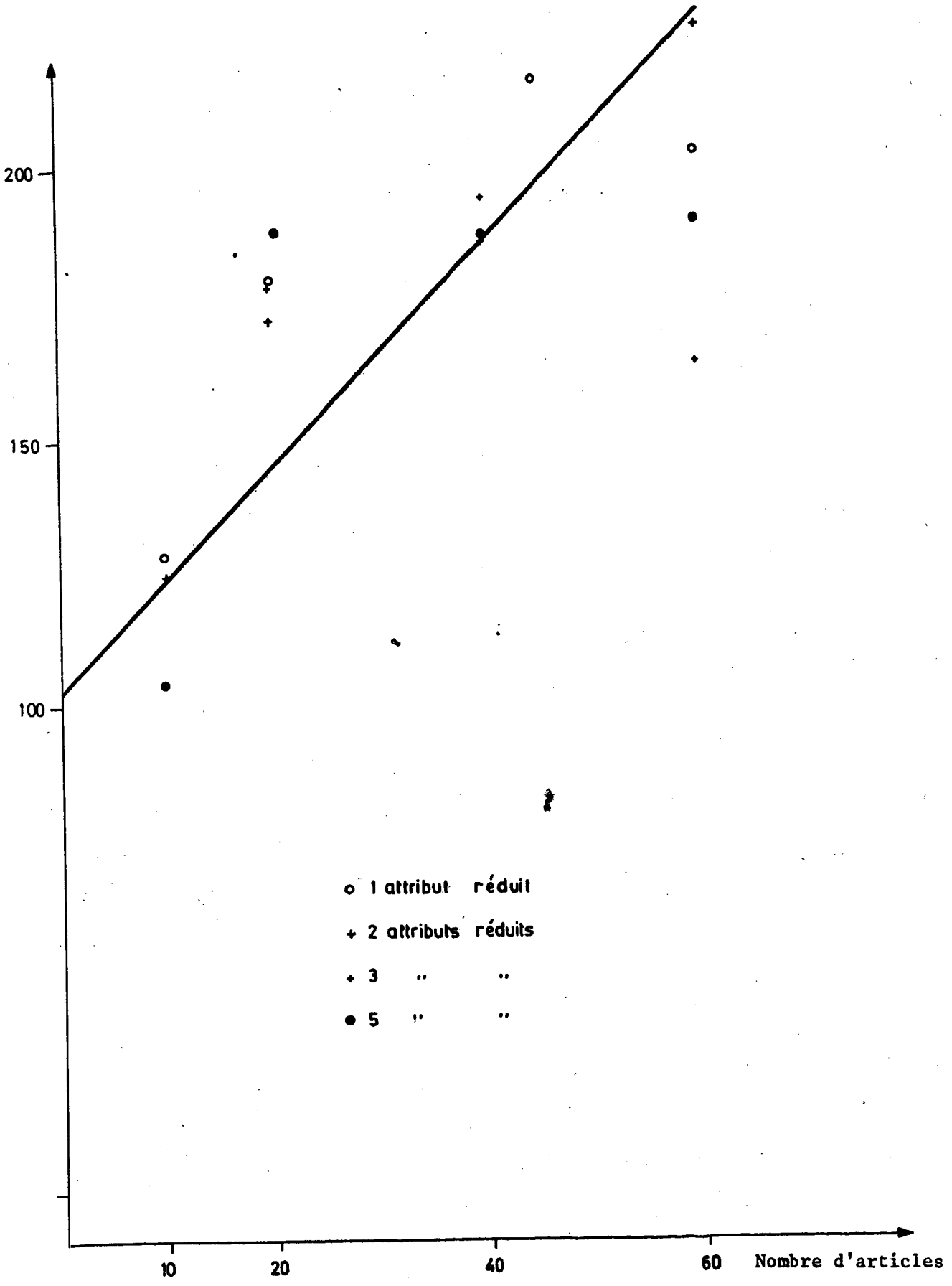


Figure 27 - Projections

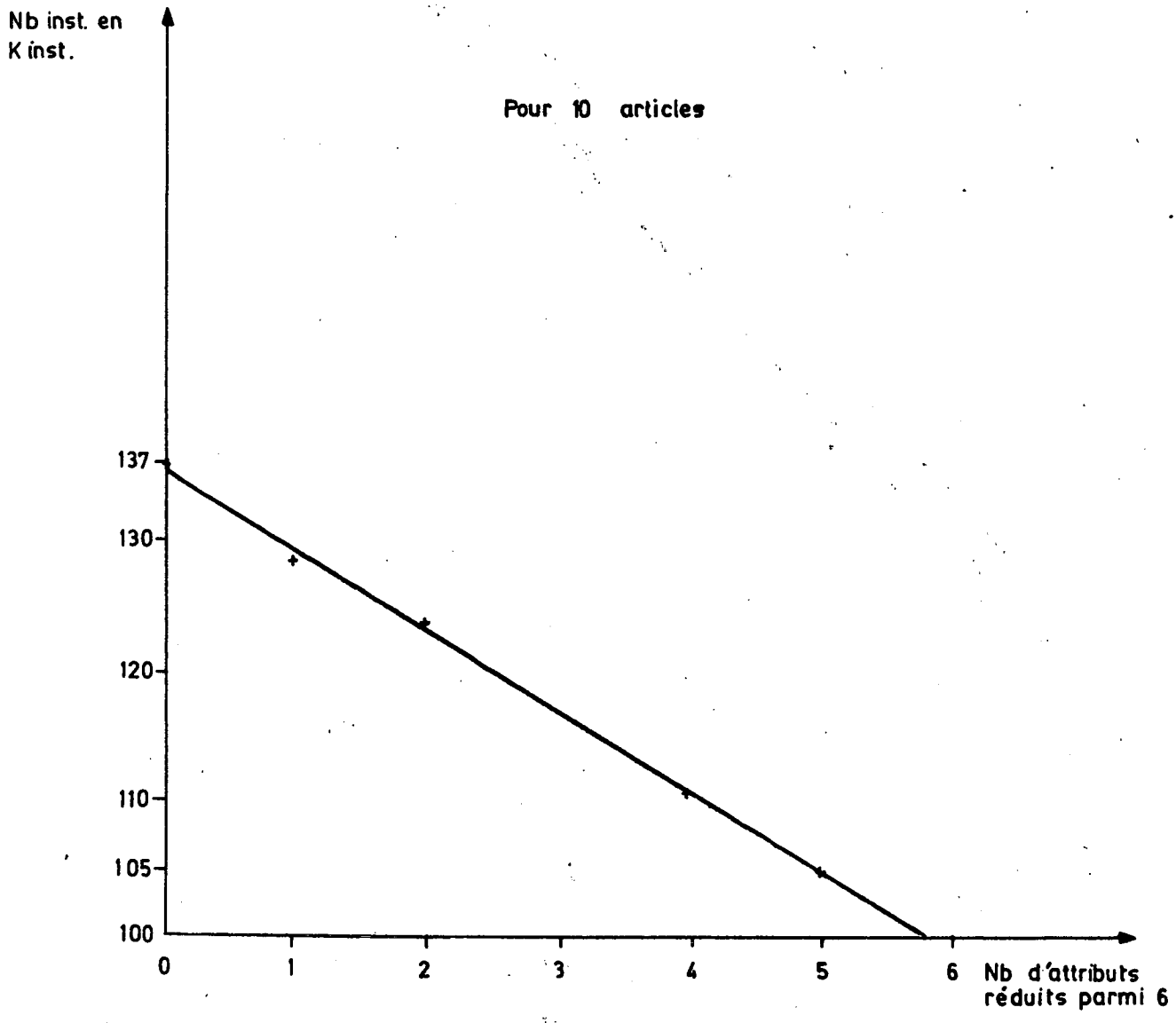


Figure 28 - Coût de la projection de 10 avec différentes projections

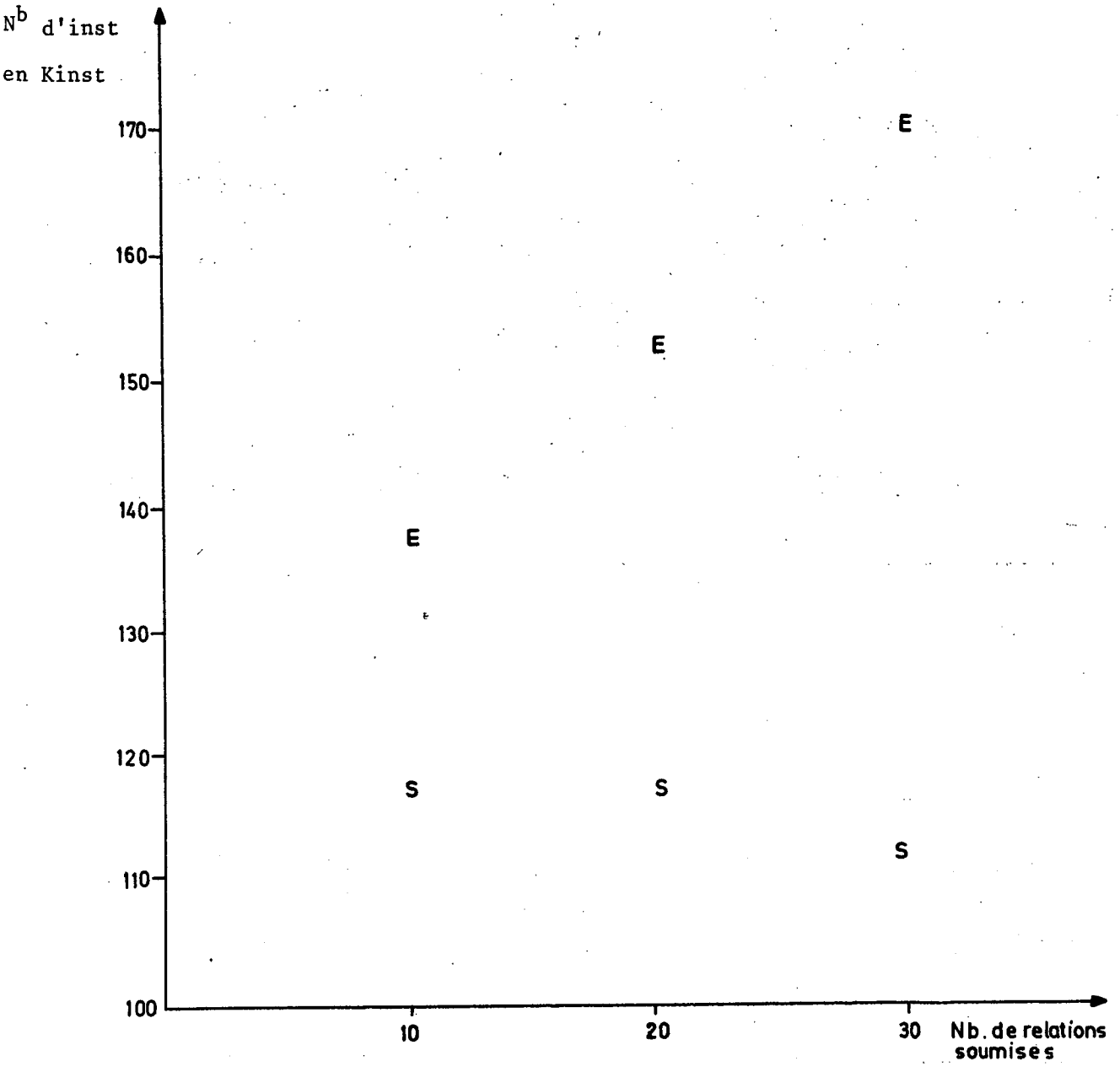


Figure 29 _ Coût des restrictions

8.5. - Conclusion

Le tableau 7-bis réalise un résumé des résultats précédemment détaillés. Il y apparaît un coût important dû à l'interprétation du langage Pivot. Ces mesures venant au sommet des couches précédentes est probablement la plus sujette à caution. Elle permet néanmoins d'avoir une idée des coûts élémentaires des opérateurs relationnels.

Union	91 000 + n * 700
Jointure	30 000 + Succès n * 1560 Echec n * 940
Projection	10 000 + n * 2000
Restriction	Non mesuré

Tableau 7-bis : Principaux résultats sur SILOE

9. EXEMPLE DE TRANSACTION

Nous allons maintenant regarder le coût complet d'une transaction afin d'évaluer comment sont réparties les différents composants : SER, SCORE, SILOE. Bien sûr nous ne pouvons avoir qu'une idée approximative pour un type de transaction. Le lecteur qui souhaiterait connaître le coût d'une transaction dont le profil est différent fera lui-même la démarche qui suit pour sa transaction.

Nous allons prendre une transaction utilisée pour les démonstrations mises au point par C. STANGRET. Nous verrons d'abord une transaction qui ne fait que des consultations puis une transaction opérant des mises à jour. Pour aider à la compréhension, nous allons d'abord décrire la base de démonstration.

9.1. Schéma d'implantation de la base de démonstration :

Description de la base de données répartie :

La base de démonstration contient des informations sur des stations balnéaires et de sports d'hiver. Elle est décrite par les trois relations suivantes :
- STATION (n° stat, nom stat, altitude, région, gare, n°hôtel, n°activités)
- HOTEL (n° hôtel, nom hôtel, téléphone, nb chambres, prix moyen)
- ACTIVITE (n° activité, typ. activité, desc. activité, tarif, contact, zone).

Description de la répartition des données :

Cette description, faite par l'administrateur des données, permet au système de "localiser" les requêtes portant sur la base de données conceptuelle précédente (les utilisateurs n'ont pas à connaître cette répartition physique).

Les "fichiers" (ou relations) STATION, HOTEL, ACTIVITE sont découpés horizontalement et verticalement. Les partitions obtenues sont alors localisées sur les différents sites de la base répartie gérée par SIRIUS-DELTA.

9.2. Transaction en consultation seule :

Nous allons réaliser une transaction qui liste les excursions possibles avec la description des activités, les tarifs et le correspondant pour les zones de montagne. L'allure de la transaction est la suivante :

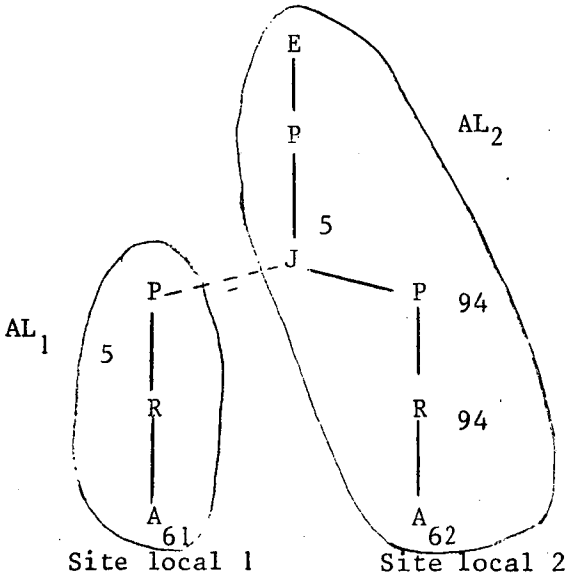
```
début - trans
    lister activité avec zone = "montagne" et avec type activité =
    "excursions", desc. activité, tarif, contact
fin - trans
```

.../...

La figure 28 décrit le schéma de notre requête, les actions locales existantes, les ressources requises (verrous, variables de synchro et FDT) et le nombre d'objets transférés. Le schéma d'exécution fait immédiatement apparaître le parallélisme possible entre les sites locaux.

Etudions maintenant les coûts dûs à chaque couche grâce aux résultats obtenus précédemment. Nous les reporterons sur la figure 29, couche par couche et pour chaque site.

"Lister activité avec zone = "mont" et avec typ. activité = "exécution" desc. activité, tarif, contact".



2 actions locales

1 VS entre AL1 et AL2
1 ET pour AL1
1 CA pour AL2
1 FDT produit par AL1
1 FDT consommé par AL2

5 articles transférés du site bleu vers le site jaune

AL1 : 1 verrou lecture
AL2 : 1 verrou lecture

Figure 28. : Schéma et ressources de la transaction

9.2.1. Coûts dûs à SER -

Le PEX soumis à SER soutient deux AL grâce à la figure 6. Nous déduisons que le coût est de 85 000 inst sur le site global et 45 000 instructions sur chaque site local pour la mise en place des AL.

L'AL_1 active une VS pour lancer l'AL_2 lorsqu'elle a elle-même débuté. De la figure 10 nous pouvons extraire les coûts respectifs :

38 000 inst pour AL_1
7 200 inst pour AL_2

.../...

Il y a en outre production d'un FDT par $1'AL_1$ et consommation de cet FDT par $1'AL_2$.

AL_1	62 250 inst
AL_2	45 000 inst

Il apparaît donc que le coût global de SER est 85 000 inst sur le site global , 145 750 inst pour $1'AL_1$ et 97 200 inst pour $1'AL_2$. Les instructions pour $1'AL_1$ et $1'AL_2$ pouvant s'exécuter en partie en parallèle. Il y a au total 287 450 instructions exécutées.

Nous comptablilisons le coût des transferts d'article de FDT comme étant propre à l'application.

9 - 2 - 2 Coût de SCORE

La transaction pose deux verrous tous les deux en lecture. Le coût des mécanismes mis en jeu par SCORE est donc 110 000 instructions sur le site global pour l'installation des tables et la prise du ticket plus 22 400 instructions puisqu'il y a deux sites atteints pour le mécanisme de validation finale (fig. 17).

Sur les sites locaux le mécanisme de pose d'un verrou et validation finale coûtent selon la figure 18, 120 000 inst.

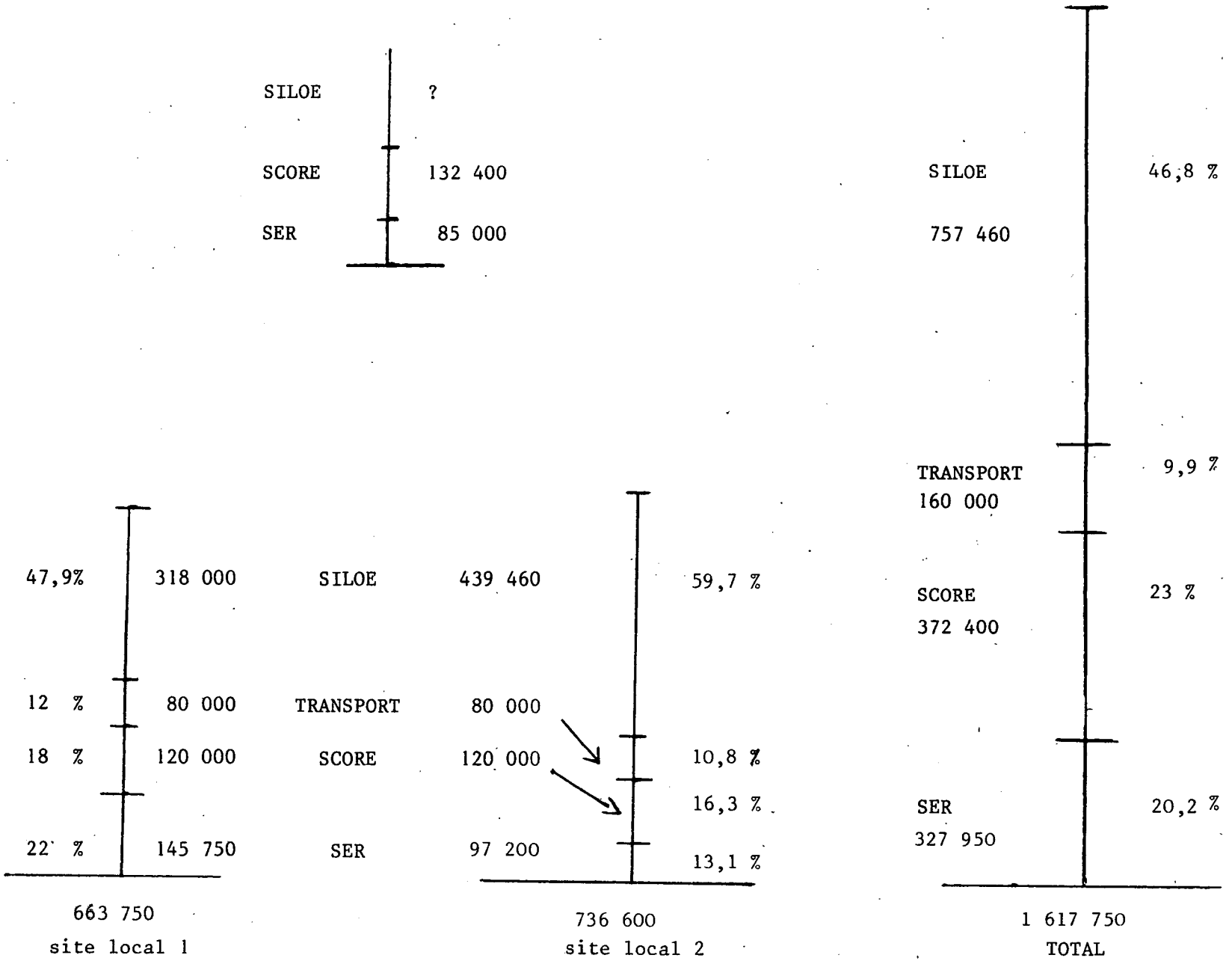
Il n'y a pas de prise d'image avant.
Le coût global de SCORE est de 372 400 instructions.

9 - 2 - 3 Coût de SILOE

Les coûts exprimés correspondent ici au travail effectivement demandé par l'utilisateur. Le seul coût spécifiquement distribué est celui qui correspond au transfert des 5 articles de FDT de $1'AL_2$. La figure 16 nous montre que ce coût est de l'ordre de 80 000 instructions pour les deux sites.

Le coût de l'analyse de la requête et de la production du PEX n'a pas pu être analysé avec le moniteur matériel (faute de temps). Ceci est regrettable le lecteur conservera donc a l'esprit qu'il manque ce coût partiellement dû au système réparti (production du PEX) et partiellement dû à l'analyse de la requête (coût existant dans tout système).

Fig.29 : Poids respectif des différentes couches



Pour évaluer le coût de l'exécution nous allons reprendre le graphe d'exécution de la figure 28 en y portant le nombre de tuples lus et consommés. Les parcelles A_{61} et A_{62} contiennent chacune 94 tuples. La partie inférieure des deux AL est identique. La restriction et la projection sont faites par une seule primitive du langage FRANCAIS, Lister. Nous assimilerons le coût d'accès et le coût de traitement de chaque tuple à celui observé pour une projection soit 2000 instructions. Nous aurons donc 94 * 2000 inst =188 000 inst. Le coût initial d'interprétation de la sous requête est d'environ 130 000 inst, un peu majoré car dans le cas d'un fichier local il y a un peu moins de travail à faire pour des FDT comme sur la figure 26.

L'AL, produit cinq tuples à transferer qui coûtent 80 000 inst sur le site emetteur surtout sur le site recepteur.

La jointure des deux sous ensembles coûte 30 000 instructions pour s'initialiser plus 7800 inst pour les 5 jointures réussies et 89 * 940 inst soit 83 000 instructions pour les échecs.

Il apparaît que le coût global de l'exécution de la question pour SILOE est de 757 460 instructions.

9 - 2 - 4 Conclusion

Nous avons reporté sur la figure 29 les coûts relatifs à chaque couche lorsque nous les connaissions pour la requête. Il y apparaît au premier regard que les coûts propres à l'exécution de la requête sont prépondérants par rapport aux coûts systèmes; et ce, même pour une requête simple sur un très petit fichier. Rappelons que le coût d'exécution de l'application dépend essentiellement du nombre de tuples présents dans les parcelles initiales. Le coût est de 47% du coût global de la requête et atteint 60% sur $1'AL_2$ qui a un peu plus de travail à faire que $1'AL_1$.

On remarquera le coût élevé 10% de transfert d'article de FDT et ce, malgré leur tout petit nombre. Optimiser le nombre de transferts est donc bien une fonction essentielle dans un SGBDR.

Le coût global de SCORE représente 23% du nombre total d'instructions réparti sur les trois sites. Sur les deux sites locaux, il atteint 18%. Le coût reste donc modéré. Néanmoins, sur cette requête, il n'y a pas mise en oeuvre du mécanisme de validation à deux phases puisqu'il n'y a pas d'écriture.

Le coût global de SER est de 20,2 % et ce, bien que presque tous ses mécanismes soient mis en jeu.

Malgré toutes les limites qu'il faut apporter à ce résultat, on constate que sur un requête simple le SGBDR consacre plus de la moitié des instructions à exécuter la requête et non à travailler pour la mise en place du système réparti.

9 - 3 Transaction avec mise à jour

Soit la transaction suivante :

```
Début trans
Modifier hôtel avec nom-hôtel = "Novotel"
     prix moyen
Fin trans
```

La fig.30 donne l'arbre d'exécution de cette transaction. Nous remarquons qu'elle est composée de 4 AL. Les actions locales 1 et 2 réalisent la sélection des tuples concernés. Lorsqu'elles ont fini une expression de terminaison de l'AL_2 autorisent le lancement des AL_3 et AL_4. Le parallelisme ici est donc limité aux AL_1 et AL_2 puis aux AL_3 et AL_4. Nous allons reprendre la même méthode que précédemment.

9 - 3 - 1 Coût SER

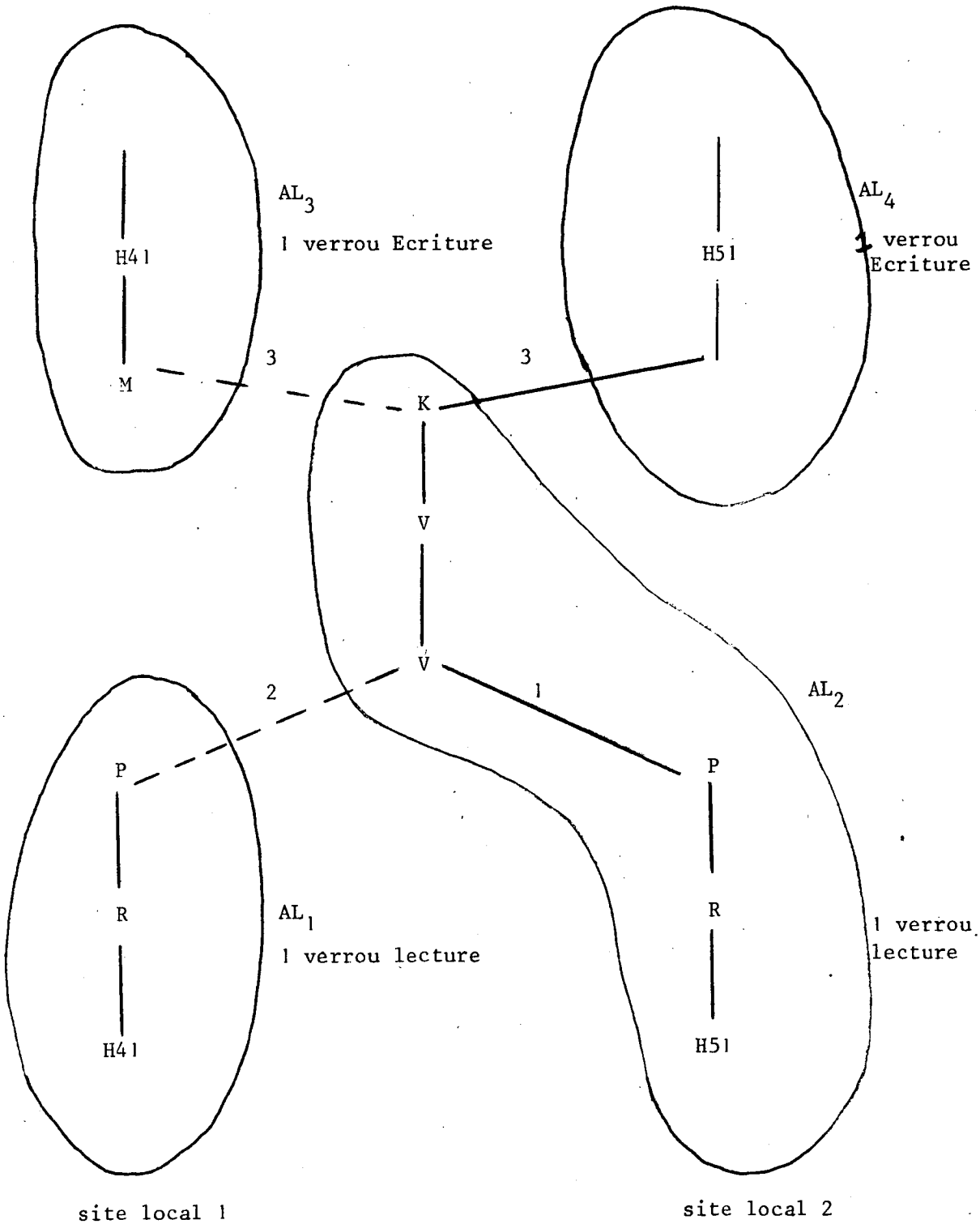
Le PEX soumis à SER contient 4 AL, son coût sur le site global est donc (fig.6) de 160 000 inst. Chacun des deux sites locaux impliqué reçoit 2 AL, il y a donc 90 000 inst.

L'AL_1 active une VS pour l'AL_2 dès qu'elle a elle même débuté. Les coûts sur la fig.10 sont donc AL_1 38 000 inst, AL_2 7 200 inst.

L'AL_2 active une VS sur les AL_3 et AL_4 dès qu'elle a terminé Nous déduisons de la fig.10 les coûts suivants :

AL_2	44 000 inst
AL_3 et AL_4	7 200 inst

Modifier Hôtel avec nom-hôtel = "Novotel" prix moyen



1 VS entre AL_1 et AL_2, ET pour AL_1, CA pour AL_2

1 FDT : produit par AL_1, consommé par AL_2

1 VS entre AL_2 et AL_3 et 4, ET pour AL_2, CA pour AL_3 et AL_4

1 FDT : produit par AL_2, consommé par AL_3 et par AL_4

fig. 30 : PEX d'une requête avec modification

Il y a, en outre, production d'un FDT par l'AL_1 et consommation par l'AL_2.

AL_1	62 000 inst
AL_2	45 000 inst

D'autre part, production d'un FDT par l'AL_2 qui le diffuse aux AL_3 et AL_4.

AL_2	62 750 inst
AL_3 et AL_4	54 000 inst

Nous constatons que l'AL_2 produit et consomme un FDT. Nous avons vu fig.13 que dans ce cas les coûts d'initialisation ne sont plus les mêmes. Nous prendrons donc comme coût celui de la mesure 1 de la fig.13, soit 69 000 inst.

Il ressort donc que le coût sur le site global de SER est de 160 000 instructions, sur le site local 1, 242 950 et sur le site local 2, 262 400. Les deux derniers coûts étant exécutés principalement en parallèle.

9 - 3 - 2 Coût SCORE

La transaction pose au total 4 verrous qui sont en fait 2 verrous en lecture sur les parcelles 448 et 451 par les AL_1 et AL_2 puis en Ecriture sur ces mêmes parcelles par les AL_3 et AL_4. Nous allons donc ici mettre en oeuvre tous les mécanismes de base de SCORE.

Il y a d'abord sur le site global 110 000 instructions puis 22 600 instructions dûes à une validation finale impliquant deux sites.

Sur les sites locaux nous avons la pose de 2 verrous par site, soit 60 000 instructions (cf fig.19), plus 90 000 instructions pour la validation finale.

Il y a par contre, une prise de 3 images_avant sur chaque site local qui coûte environ 30 000 inst.

Il y a donc un coût de SCORE sur le site global de 132 600 instructions et sur chaque site local de 180 000 inst. Cette dernière partie étant exécutée en parallèle.

9 - 3 - 4 SILOE

Les coûts exprimés ici correspondent au travail explicitement demandé par la requête de l'utilisateur. Le coût spécifiquement distribué est celui dû aux transferts de FDT. Il y a 3 articles transférés de AL_1 vers AL_2 puis de AL_2 vers AL_3. La fig.16 nous permet d'évaluer à 75 000 instructions chacun de ces transferts qui sont exécutés l'un après l'autre dans ce PEX. Il y a 150 000 instructions exécutées pour les transferts d'articles de FDT sur chaque site.

Pour évaluer le coût de l'exécution nous allons reprendre le graphe d'exécution de la fig.30 en y portant le nombre de tuples accédés dans chaque parcelle. Les AL_1 et AL_2 réalisent toutes les deux une restriction puis une projection, en fait un "Lister" Français. En nous basant sur 2 000 instructions par tuple nous obtenons :

AL_1	2 000 * 79	= 158 000
AL_2	2 000 * 128	= 256 000

L'AL_2 réalise l'union des tuples selectionnés, il y en a trois au total.

91 000 + 3*700 = 93 100 inst

Pour chaque AL nous devons ajouter 130 000 instructions par AL pour interpréter le graphe de la sous-requête.

Les AL_3 et AL_4 reçoivent le nom des hôtels dans lesquels il faut modifier le prix. Le coût est donc ici très simple 130 000 inst pour interpréter le graphe de la sous-requête puis 3 écritures normales R2000 (fig.20).

3*760 = 2 280 inst

Nous obtenons donc pour SILOE un coût global de 1 031 660 inst sur les sites 1 et 2. Décomposé en 420 280 instructions pour les AL_1 et AL_3 et 611 380 inst pour les AL_2 et AL_4. L'AL_2 à elle seule représente 479 100 instructions. Elle est la plus coûteuse des 4 AL ce qui était évident en regardant le graphe d'exécution (fig.30).

9 - 3 - 5 Conclusion pour une transaction en mise à jour

Nous avons reporté sur la fig.31 les coûts relatifs de chaque couche pour cette transaction composée d'une requête très simple sur une très petite base. Nous observons ici que globalement 41,4% des instructions sont utilisées pour SILOE et, en outre, 12% sont dû aux transferts des tuples. Il y a donc une majorité d'instructions pour effectuer le travail de la requête. Côté système on remarque que le poids de SER

augmente par rapport à l'exemple précédent du fait qu'il y a 4 AL au lieu de 2. Par contre, le poids de SCORE ne suit pas la même augmentation bien que les mise à jour soient exécutées.

On remarquera que la grande majorité du travail est exécutée en parallele sur les deux sites locaux. Par contre, la charge du site global augmente peu.

9 - 4 Conclusion

Le lecteur remarquera que le calcul du coût d'une transaction est immédiat dès que l'on connaît le PEX de chacune des étapes. Une automatisation pourrait être faite très simplement en rentrant dans une base de donnée :

- l'ensemble des résultats de mesure décrits dans ce papier
- les caractéristiques de la base de donnée, nombre de tuples dans chaque parcelle, etc...

Il s'agit du schéma interne global et de son complément quantitatif. Il faut, en outre, disposer de la partie génération de PEX faite par SILOE global. A partir de ces informations le calcul est automatique.

Le lecteur aura alors à sa disposition deux résultats essentiels :

- le parallelisme obtenu en regardant le graphe d'exécution du PEX
- le coût de chaque AL, donc, comment se passe effectivement le parallelisme et finalement le coût global. La présentation des résultats sous la forme des schémas parallèls utilisés sur les fig.29 et 31 permet de voir de parallelisme effectif.

Le lecteur remarquera, néanmoins, que si la base de donnée mesure est à faire une seule fois, par contre la description du schéma interne global et de la répartition est à la responsabilité de l'utilisateur qui veut évaluer son projet de BDR.

Les mesures précédentes peuvent être utilisées pour l'évaluation de bases futures comme nous venons de le montrer. Elles peuvent aussi être utilisées utilement pour valider des modèles. Le lecteur gardera, néanmoins, à l'esprit que les extrapolations faites à partir des mesures sur un prototype limité sont parfois délicates. Elles sont toutefois possibles et il serait très intéressant par exemple d'étudier le coût d'exécution d'une très grosse base partitionnée sur un grand nombre de processeurs. Dans ce cas, il est évident que le nombre total d'instructions sera plus élevé qu'en centralisé mais le parallelisme devrait beaucoup diminuer le temps de réponse. On peut dire pratiquement que la hauteur de la colonne SILOE dans les schémas 29 et 31 représente le temps de réponse en centralisé et par contre, en distribué, la somme de la plus haute colonne des sites locaux et du site global approche (à la vitesse du réseau près) le temps de réponse en distribué(sur notre prototype avec seulement deux sites locaux).Il y a déjà pratiquement équilibre. On imagine bien que sur des exemples plus complexes pour notre prototype la balance penchera en faveur du distribué. Si, en plus, on augmente le nombre de sites locaux et le partitionnement de la base le temps de réponse devrait alors être franchement en faveur du système réparti.

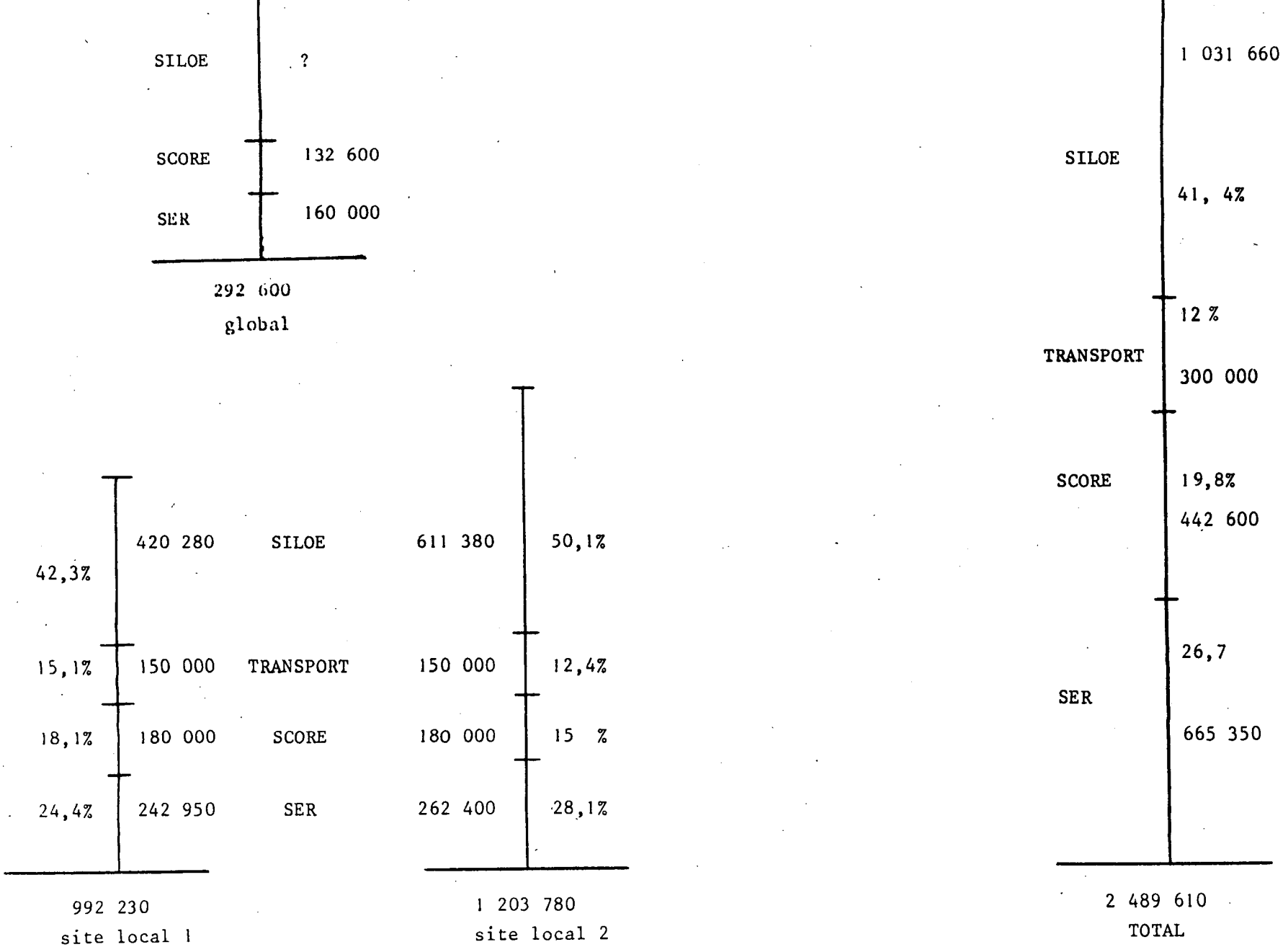


fig. 31 : Poids respectif des différentes couches

BIBLIOGRAPHIE

[GLOR80] A.M. GLORIEUX : Décomposition d'une requête dans SIRIUS-DELTA : Types de Répartition et Méthodologie, Rapport Interne INRIA LOC-I-001 - juin 80.

[INT 1] Manuel de référence R2000. Document Intertechnique.

[LEBI79] J. LE BIHAN, C. ESCULIER, G. LE LANN, L. TREILLE. "SIRIUS-DELTA : un prototype de système de gestion de bases de données réparties" SCH-I-072 INRIA SIRIUS.

[LEL 79] G. LE LANN : Consistency and concurrency control in distributed data base systems. EEC course "Distributed data base" Sheffield (U.K). (Doc. SIRIUS INT-I-007).

[LEL1-79] G. LE LANN : Les problèmes de signalisation dans les systèmes informatiques à contrôle réparti, SYN-I-008 INRIA SIRIUS.

[LEL 80] G. LE LANN : An experimental fault-tolerant distributed data-sharing system, CTR-I-025 INRIA SIRIUS.

[MESI006] P. ROLIN : Eléments de mesure sur le niveau transport dans SIRIUS-DELTA. Document SIRIUS MES-I-006.

[ROL 79] B. CANET, B. DECOUTY, G. MICHEL, P. ROLIN, C. WAGNER : Campagnes d'évaluation de performances réalisées à l'aide du système Caméléon, Publication IRISA n° 118, Université de Rennes, 35031 Rennes.

[SED 81] S. SEDILLOT : Signalisation des événements, mise en oeuvre du séquenceur circulant, Doc. SIRIUS SYN-I-008.

[SDD1-79] SDD1 technical report Jan 79 - Section 2 - pp. 107-143.

[STAN81] A.M. GLORIEUX, C. STANGRET : L'hétérogénéité dans le SGBDR SIRIUS-DELTA - CIL-81 Barcelone (Proceedings).

[STO 79] M. STONEBRAKER : Concurrency control and consistency of multiple copies of data in distributed INGRES, IEEE transactions on software engineering, May 79.

[TRE 79] L. TREILLE, G. SERGEAN : SER : système d'exécution à contrôle décentralisé, Bibl SIRIUS XEC-I-009.

Imprimé en France

par

l'Institut National de Recherche en Informatique et en Automatique